VECTOR FIELD MANIPULATIONS IN 3-D TIME-VARYING FLOW DATA COMPRESSION AND IMAGE/VIDEO EDITING

BY

HUI FANG

B.S., Nanjing University, 1998
M.S., University of Illinois at Urbana-Champaign, 2002

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2006

Urbana, Illinois

VECTOR FIELD MANIPULATIONS IN 3-D TIME-VARYING FLOW DATA

COMPRESSION AND IMAGE/VIDEO EDITING

Hui Fang, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 2006
John C. Hart, Advisor

Both the measurement and simulation of unsteady flow yield huge datasets of time-varying volumetric vector fields. Flow dynamics researchers face the very difficult task of finding, extracting and analyzing important flow features (e.g. vortices, shocks) buried in these massive datasets. An MPEG-like method is developed for the compressed transmission of time-varying 3-D flow datasets that emphasizes on feature preservation. Key frames of the flow motion are compressed using a harmonic analysis of the flow. A novel bi-directional advection model is then used to approximate intermediate frames. Key features like vortical structures and shocks lost in compression are reconstructed during visualization. Our algorithm has a much larger compression rate compared to compressing each frame individually and preserves interesting or important flow features in a large unsteady flow dataset.

Textureshop is an image editing system that applies texture onto a surface in a photograph. Shape from shading is used to approximate normal field on the surface. Then texture synthesis is applied on that surface with texture coordinates deformed by the recovered normal field. The result is a texture that follows the undulation of the surface.

Rototexture is a normal based video editing system that allows a user to apply a time-coherent texture to a surface depicted in the raw video from a single uncalibrated camera, including the surface texture mapping of a texture image and the surface texture synthesis from a texture swatch.Our system avoids the construction of a 3-D shape model and instead uses the recovered normal field to deform the texture so that it plausibly adheres to the undulations of the depicted surface. The texture mapping method uses the non-linear least-squares optimization of a spring model to control the behavior of the texture image as it

is deformed to match the evolving normal field through the video. The texture synthesis method uses a coarse optical flow to advect clusters of pixels corresponding to patches of similarly oriented surface points. These clusters are organized into a minimum advection tree to account for the dynamic visibility of clusters. We take a rather crude approach to normal recovering and optical flow estimation, yet the results are robust and plausible for nearly diffuse surfaces such as faces and t-shirts.

Morphing is a common practice in digital photograph editing, but morphing does not replace the detail lost where the image is enlarged. We propose an image editing system that decouples feature position from pixel color generation to achieve a morph that preserves texture detail and orientation near the dragged silhouette, synthesized using the original image as an anisotropic texture. We introduce a new distortion to patch-based texture synthesis that aligns texture features with image features. A dense correspondence field between source and target images generated by the control curves can then guide texture synthesis.

Acknowledgements

I’d like to thank my adviser John C. Hart, who is always supportive and allows me a lot of freedom in pursuing research topics that I found interesting. His help both in research and in English writing made all my publications possible. I’d also like to thank my committee, David Forsyth, Michael Garland, Thomas Huang, Xiangmin (Jim) Jiao and Yizhou Yu for their advices. I also want to thank everyone who helped me during my research, especially those who offered their faces for unnatural texture mapping and dramatic deformations.

Thanks to my wife, Wei, and my family far far away in China, for the endless support during all these years. Special thanks to my daughter Amelie. You chose to be born 59 hours after, not before, my defense.

My research was supported in part by the NSF under the ITR OCI-0121288.

Contents

Chapter 1

Introduction

This thesis uses vector field manipulations, including bi-directional advection and seamless local deformation, to build applications in 3-D time-varying flowdata compression and image/video editing.

1.1 FlowPEG

Computational fluid dynamics (CFD) simulations and fluid dynamics experiments are capable of generating huge terabyte-scale time-varying flow datasets. Due in part to exponential growth in processing power, these datasets are being generated at a faster pace than scientists can currently cope with, requiring the warehousing of the flood of data for later examination. While some are developing efficient visualization tools aimed at personal computers, the delivery of large datasets from a server to the PC remains problematic.

In order to cope with the massive amount of data, scientists employ a *triage* approach, quickly examining datasets for interesting features such as vortices or shocks. While some automatic methods for flow feature detection exist, they are not yet foolproof and still require human processing to avoid false positives. The triage of datasets can occur quickly, and does not require absolute data fidelity.

We propose FlowPEG, a method for the lossy compression of the time-varying 3-D

vector fields resulting from unsteady flow simulations and measurements. Lossy compression of these datasets makes their transmission faster, and allows scientists to store larger datasets locally on personal computers. One problem with lossy compression is that it reduces the fidelity of the data, bringing its scientific accuracy into question. We introduce a scheme that the server specifies and transmits important flow features that a client should reconstruct during visualization. Although such scheme is also lossy, the results plausibly convey the underlying features.

The result is a tool for scientists to quickly examine large flow datasets for interesting features. Once an interesting feature is detected, the scientist can then request a full-fidelity lossless version of the data to perform a more thorough examination of the phenomena to draw meaningful and justifiable conclusions.

Generally speaking, the more is known about the nature of the data to be compressed, the better the compression rate could be. Then it is natural to utilize the physics of fluid while compressing flow datasets. Our compression method predicts flow motion from key frames by performing a quick flow simulation on an initial vector field. An ideal predictor for a simulated dataset would match the process used to create the flow data in the first place. However, sophisticated flow simulations use computational power that exceeds the capabilities of the PC found on a typical scientists desktop. Moreover, the measurement of real world data defies exact prediction from computational simulations. Hence, our flow codec is instead based on a simpler predictor by only applying bidirectional advection that can be executed in real-time, or at least at interactive rates, on a standard desktop personal computer.

1.2 Textureshop

Texture synthesis has revolutionized the construction of texture maps and the application of texture to surfaces. Given a texture swatch, it uses a machine learning process to plausibly

extrapolate its pattern, extending the texture features across an image plane or the surface of a geometric model [1; 2].

Textureshop is a novel application of texture synthesis in photograph editing. Various commercial software packages for photograph editing exist, but they work in the image space and do not offer a direct method to apply texture to the surface of a photographed object, and using these tools for the convincing texturing of an undulating surface requires both time and skill.

Shape-from-shading techniques can recover a height field from the image of a shaded surface, and existing texture synthesis techniques could be applied to a recovered mesh. However, such reconstruction is complex, expensive and inaccurate, particularly in the presence of noise or an unknown reflectance map.

We overcome these limitations by creating small pixel patches, clustered by similar recovered normals. We perform texture synthesis on these patches, distorting pixel positions into a local parameterization to account for patch orientation and displacement. These patches are further distorted to match the features of neighboring patches. This patchwork of feature-aligned foreshortened textured pixel clusters gives the illusion that the texture is applied to the photographed surface. We furthermore apply shape-from-shading to the texture to support displacement mapping and normal transfer (embossing).

1.3 Rototexture

It is a natural extension to apply Textureshop on a surface in a video sequence. The major issue we need to deal with is the inconsistency of generated texture between frames. Optical flow captures the motion between two frames. But a pixel-wise accurate optical flow is generally difficult to achieve for a video sequence. Feature tracking finds corespondent features between two frames. Such features usually offer most reliable vectors for optical flow and they are interpolated to form a smooth optical flow. We allow non-linear advection

of the texture to similar frames using a Minimum Advection Tree.

Such optical flow will still not be pixel-wise accurate. If we advect generated texture for the first frame with it, texture patterns will quickly be distorted and show the inaccuracy in the optical flow. So instead, we only use it to advect clusters on the surface, and leave details within each cluster to be decided with local normals. Texture of neighboring clusters is merged seamlessly with graphcut. 3-D graphcut is applied to achieve the smoothness in the seam positions between frames.

We also overcome the cluster size limitation of Textureshop by using a spring network. With that we may paste a whole image onto a surface in a photo or a video clip.

1.4 Detail Preserving Image Morphing

Large morphing in digital photograph editing normally leaves the image details distorted. Instead, patch based texture synthesis can be used to fill the details after a morphing. But there are two questions that need to be answered for such an approach: where to sample source texture, and how to maintain the details along arbitrarily deformed feature lines.

Our morphing tool allows a user to intuitively select and deform several control lines in an image. The resulting image is morphed based on the deformed control lines. We proposed an additional distortion to patch based texture synthesis. This distortion bends texture patches along deformed feature lines. A deformation field is also generated from control lines to determine where to sample texture in the source image.

Chapter 2

Compress and Advection of Frames in FlowPEG

2.1 Introduction

Current computer power enables computational fluid dynamics (CFD) simulations and fluid dynamics experiments to generate high resolution results equivalent to terabyte-size time-varying flow fields. These datasets are generated at a faster pace than scientists can currently cope with, requiring the warehousing of the flood of data for later examination. While some are developing efficient visualization tools aimed at personal computers, the delivery of large datasets from a server to the PC remains problematic.

In order to cope with the massive amount of data, scientists employ a *triage* approach, quickly examining datasets for interesting features such as vortices or shocks. While some automatic methods for flow feature detection exist, they are not yet foolproof and still require human processing to avoid false positives. This triage can occur quickly, and does not require absolute data fidelity.

Existing compression methods that can be extended to volumetric data compression are reviewed in Section 5.2, but these methods do not easily generalize to the challenging obstacles presented by unstructured grids supporting complex time-varying physical

characteristics. We thus propose a new method called FlowPEG designed specifically for the lossy compression of the time-varying 3-D vector fields resulting from unsteady flow simulations and measurements on unstructured grids.

Lossy compression of these datasets makes their transmission faster, and allows scientists to store larger datasets locally on personal computers. Lossy compression by definition reduces the fidelity of the data, which can raise concern on the scientific accuracy of the reconstruction. Section 2.3 shows how FlowPEG overcomes this concern by introducing a scheme with a physics-based MPEG-like encoding, augmented with a second side-band channel. This channel contains information about key flow features detected from the original full-fidelity dataset. This ensures the client that important flow features are preserved. The encoding of these features is also lossy, but the results plausibly convey their configuration.

The codec is not intended to support detailed inspection and analysis, but instead to provide a quick (and dirty) overview of the dataset for a feature of interest. Once an interesting feature is detected, the scientist can then request a full-fidelity lossless version of the data to perform a more thorough examination of the phenomena to draw meaningful and scientifically justifiable conclusions.

The procedure of data compression we propose is as follows. A photographic image is recorded on a regular rectilinear grid, and its compression typically relies on the harmonic analysis provided by Fourier transform, e.g. the discrete cosine transform employed by JPEG. The interesting parts of a flow often occur at different scales and in different locations, so fluid flow datasets are commonly recorded on non-uniform, irregular or even unstructured grids. Section 2.4 describes how FlowPEG performs a harmonic analysis on the multi-dimensional flow signal over such a grid by examining the eigen-structure of its graph Laplacian and transforming the dataset into a frequency spectrum, a representation more suitable for compression.

In general, the more we know about a dataset, the better we can compress it. We thus

capitalize on the physics of fluids as a model to improve the compression of flow datasets. FlowPEG predicts flow motion from key frames by performing a quick approximate flow simulation on an initial vector field, as shown in Section 2.5. An ideal predictor for a simulated dataset would match the process used to create the flow data in the first place. However, sophisticated flow simulations use computational power that exceeds the capabilities of the PC found on a typical scientists desktop. Moreover, the measurement of real world data defies exact prediction from computational simulations. Hence, our flow codec is instead based on a simpler predictor based on bidirectional advection.

This approximation of flow tends to destroy high-level flow features such as shocks and vortices, which are separated and encoded on a second channel. Chapter 3 describes how FlowPEG detects and encodes shocks and vorticity, which do not survive spectral encoding and bidirectional advection, so they may be reconstructed and reintegrated into the flow during decompression and display.

We offer a proof of the FlowPEG concept with its demonstration on two simple datasets. The first is an unstructured dataset depicting a flow through a pipe with a shock, whereas the second is a rectilinear grid dataset depicting flow along channels with many vortices. Section 6.2 reports a compression rate for the pipe flow of about 110:1, decompressed at a fidelity that is visually identical to the original, and for the channel flow a rate of about 1000:1 at a fidelity that clearly indicates some loss of data but nevertheless faithfully reproduces the presence of the vortices and turbulence. This leaves Section 6.3 to conclude that FLowPEG indeed is capable of capitalizing on the redundancies offered by physical simulation while detecting and preserving important flow features to provide a mechanism for the low-bandwidth transmission and feature-faithful browsing of large collections of flow datasets.

2.2 Previous Work

Computational fluid dynamics has been successful at simulating a wide variety of experiments in fluid mechanics and dynamics based on the Navier-Stokes equations [3]. Accurate CFD simulations are nonetheless expensive, and this application currently demands very large computers and mass storage devices.

In some cases, accuracy is less important than speed, so long as the motion is *plausible.* One realm where plausible is good enough is the computer graphics used for motion picture production and video game development, where flow need only be visually convincing. Simplified Navier-Stokes equations are used to generate the motion of incompressible fluid to for example simulate the dynamics of smoke [4; 5]. Even in simplified form, the computation of incompressible fluid flow involves solving a large linear system. Instead, we use even more simplified dynamics to achieve faster simulation speed. Static vector fields can also be simplified using a variety of recent techniques [6; 7]. These methods apply clustering to abstract features from a single vector field. A subdivision scheme has also been introduced to realistically interpolate a dense vector field from a subsampled field, or from a "sketch" [8]. Simplification and subdivision techniques combine to yield up- and down-sampling filters that could provide a multiresolution/filter-bank basis for vector field encoding.

JPEG is a standard for lossy picture compression based on the 2-D DCT [9]. A similar form of compression has been extended to compress 3-D uniform-grid volumetric scalar fields [10]. Taubin uses the eigenvectors of the Laplacian defined on mesh vertices for mesh smoothing [11]. We extended his method into a compression scheme for datasets defined on unstructured points. Other methods also exist to compress static or animated volume data. Fowler proposed a combination of differential pulse-code modulation(DPCM) and Huffman coding to losslessly compress volume data[12]. MPEG-1 is a standard for lossy compression of motion pictures using DCT and motion prediction/interpolation [13], which can also be extended to compress time varying volume data. Guthe used a wavelet based

compression scheme and an MPEG-like block matching motion compensation method to compress/decompress animated volume data for realtime visualization [14].

Researches were also done on transmitting and visualizing volume data over a narrow network channel while preserving important scientific features. Specific methods are available to extract flow features, for example, vortical structure from 3D CFD vector fields[15]. Content-based compression is used to preserve features by doing lossless compression at the region of interest[16]. Such a scheme was used in medical research to both compress background regions at a high rate and preserve important features at focus-of-attention regions[17]. Sohn achieved high compression ratios on time-varying isosurfaces and volumetric features by only encoding significant blocks in a 3d wavelet compression scheme[18].

All these methods aim at general volume data. In this paper, we try to develop a specific compression scheme suitable for flow data. Our approach replaces the MPEG-1 motion predictor with simplified advection, which is a natural predictor for fluid dynamics datasets. This choice makes our codec domain specific, but time varying vector fields that are not based on some form of fluid dynamics are few and far between, arising usually from purely mathematical studies.

2.3 FlowPEG Scheme Overview

FlowPEG utilizes a compression scheme that is similar to MPEG-1 standard. The original flow datasets contains one static vector field u_t at each integer time step t. Even though the vector field u_t is three-dimensional, we will call it a "frame" to better compare the technique to existing video compression techniques. Similar to the MPEG-1 standard, our encoder sends intra (I) frames that are transmitted independently from other frames. Predicted (P) frames are then constructed from nearby intra frames. MPEG-1's P-frames require transmission of block motion data. The P-frames in our flow data application are

predicted from a fixed equation. Hence, the data stream resembles a modified run-length encoding with alternating I-frames and P-frames.

$$IPP..PPIPP..PPIPP.. \tag{2.1}$$

However, important flow features may be lost during the advection. Additional feature data is needed at each frame for reconstruction. Such feature data comes either from additional feature Metadata or from advecting features from key frames. In a final stage the features are reconstructed and blended into the datasets. The whole FlowPEG scheme is shown in the following figure.

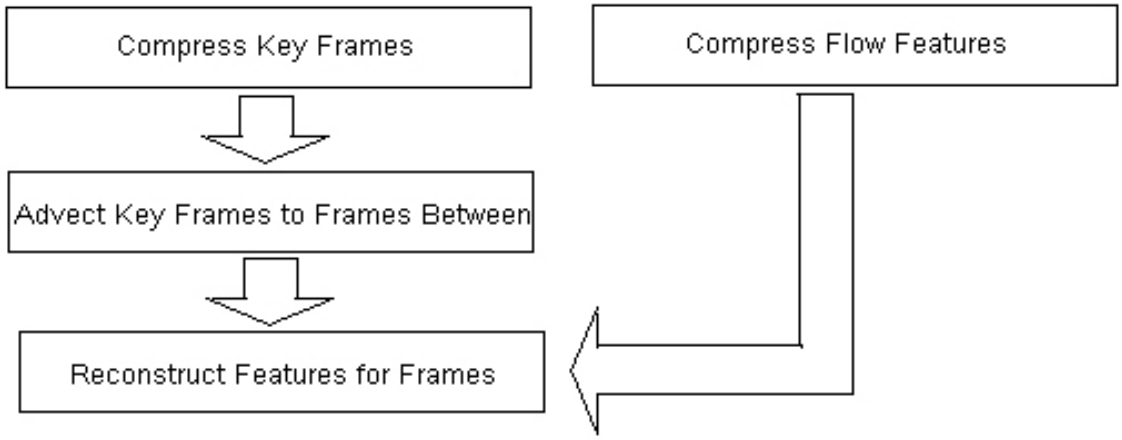


Figure 2.1: FlowPEG compression scheme.

2.4 Compression of I frames

Traditionally, 3-D scalar fields are compressed based on Fourier decomposition. The Fourier transform is well understood and there exist fast algorithms to compute it. We used an existing DCT compression method [10] to compress I-frames for structured grid datasets. Each vector component is treated as an independent scalar function which is transformed into frequency space and compressed by encoding its frequency spectrum. For large vector fields, especially in three dimensions, the data is evenly divided through domain de-

composition. Each domain is compressed separately. The DCT coefficients of the vector fields were Huffman coded in the usual way to further increase their compression rate.

However, to apply Fourier analysis requires a regular grid of samples which severely limits its applicability. In practice, many vector fields are sampled on semi-regular or even unstructured grids with highly adaptive sampling densities. Resampling these grids is prohibitively expensive both in terms of storage and due to the sampling artifacts introduced in the process. We propose a new compression scheme that can be applied to datasets defined on unstructured points. This scheme is called Laplacian eigenvectors based compression scheme.

2.4.1 Laplacian Eigenvectors Based Compression Scheme

Instead of relying on a regular grid structure we extend tools from spectral graph theory to three dimensions to allow frequency analysis on arbitrary grids. For triangulated surfaces it is well known that the discrete Laplace operator can be expressed as

$$\Delta f_i = \sum_{j \in N_i} w_{ij}(f_j - f_i), \tag{2.2}$$

where N_i is the set of vertices adjacent to vertex i and w_{ij} is a weight associated with the edge (i, j).The choice of weights depends on the application, but for surface analysis the *discrete harmonic* weights $w_{ij} = -\frac{1}{2}(\cot\alpha_{ij} + \cot\beta_{ij})$, where α_{ij} and β_{ij} are the angles opposite edge (i, j), have been shown to perform well. For a more detailed derivation we refer the reader to [19–21]. The Laplacian can be reformulated as a matrix equation

$$\Delta \vec{f} = -L\,\vec{f}, \tag{2.3}$$

where $\vec{f} = [f_1\ f_2\ \ldots\ f_n]^\mathsf{T}$ and

$$L = \begin{cases} \sum_k w_{ik} & \text{if } i = j, \\ -w_{ij} & \text{if } (i, j) \text{ is an edge of } M, \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

The eigenvectors of L, $\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_n$ form a basis of the spaces of piece-wise linear functions over the surface. In [11], Taubin shows that the associated eigenvalues $\lambda_1 = 0 \leq \lambda_2 \leq \ldots \leq \lambda_n$ correspond to frequencies of these basis functions where a larger eigenvalue indicates a higher frequency. Expressing a function as combination of Laplacian eigenvectors, therefore corresponds to a frequency decomposition of the function similar to a Fourier transform of a traditional signal. This correspondency has been used very successfully, for example, to smooth surfaces [11] by removing its high frequency components or to compress surfaces [22].

Given a volumetric mesh rather than a triangulation, one only needs a new definition of the weights w_{ij} to apply the same methods as described above. A laplacian with weights only depending on distance to other grids within a radius offers a most general way for data compression. It operates on datasets defined on any type of grid or no grid at all, thus is independent of topology. It may naturally adapt to changing sampling density. Other weight definitions generalized from mesh surface (e.g. [23]) exist if the volumetric meshes are known.

Given the weights, the compression algorithm becomes straight forward. We first segment the volume into regions containing a roughly equal number of samples. For a region, we compute the Laplace matrix and its eigen-decomposition. The total eigen vectors number equals the total grids number K. Each vector field component of a dataset $\vec{x}$ can be express as linear combination of the K eigen vectors $\vec{e}_k$:

$$\vec{x} = \sum_{k=0}^{K} c_k \cdot \vec{e}_k \tag{2.5}$$

Then we compress the resulting frequency spectrum. Since $\vec{e}_k$ are ordered from low frequency to high frequency, c_k can be compressed exactly like the zigzagged coefficients in DCT compression [9].

Since the Laplacian matrix depends only on the underlying grid and not on the vector field itself it is implicitly encoded in the underlying grid and therefore does not need to be stored. Furthermore, for time-dependent data the matrices remain the same for each time

step and therefore need to be computed only once for all time steps.

DCT compression is a special case of our compression algorithm, since cosine series are the eigen vectors of the Laplacian on regular grid.

2.4.2 Compression Example on Unstructured Points

In the most general format, the dataset $\vec{x}$ is defined on unstructured points, where the connectivity between points are unknown. In such a case, the Laplacian of each point is defined only by the distances to its neighboring points. In this example the shock datasets from section 3.3.1 is divided into uniform cubic regions. A kd-tree is constructed containing all the points within each region. For each grid p_i, its neighbors n_{ij} within radius R are found with the kd-tree. The Laplacian is defined as:

$$L(\vec{x}(p_i)) = \sum w_{ij}(x(p_i) - x(n_{ij})) \tag{2.6}$$

where $w_{ij} = \frac{1}{distance(p_i, n_{i,j})}$.

The Signal-Noise Ratio (SNR) on key frames in the shock dataset is given as blue line in Figure 2.2. In comparison, the red line shows a typical SNR curve of traditional DCT scheme on the channel turbulent dataset. The much higher SNR for shock dataset with low compression rate is partly due to the fact that the shock datasets contains less high frequency details, as can be seen from the final results. Such scheme can be efficiently applied to time-varying datasets without recalculating the eigenvectors if the position of the unstructured points remains the same over time.

2.5 Bi-directional Advecting from I-frames

In many cases of visualization such as dataset browsing, what matters to the viewer is not the L^2 norm of the error in the whole visualized field, but whether it conveys the evolution of the flow and its features. FlowPEG offers a much higher compression rate than compressing

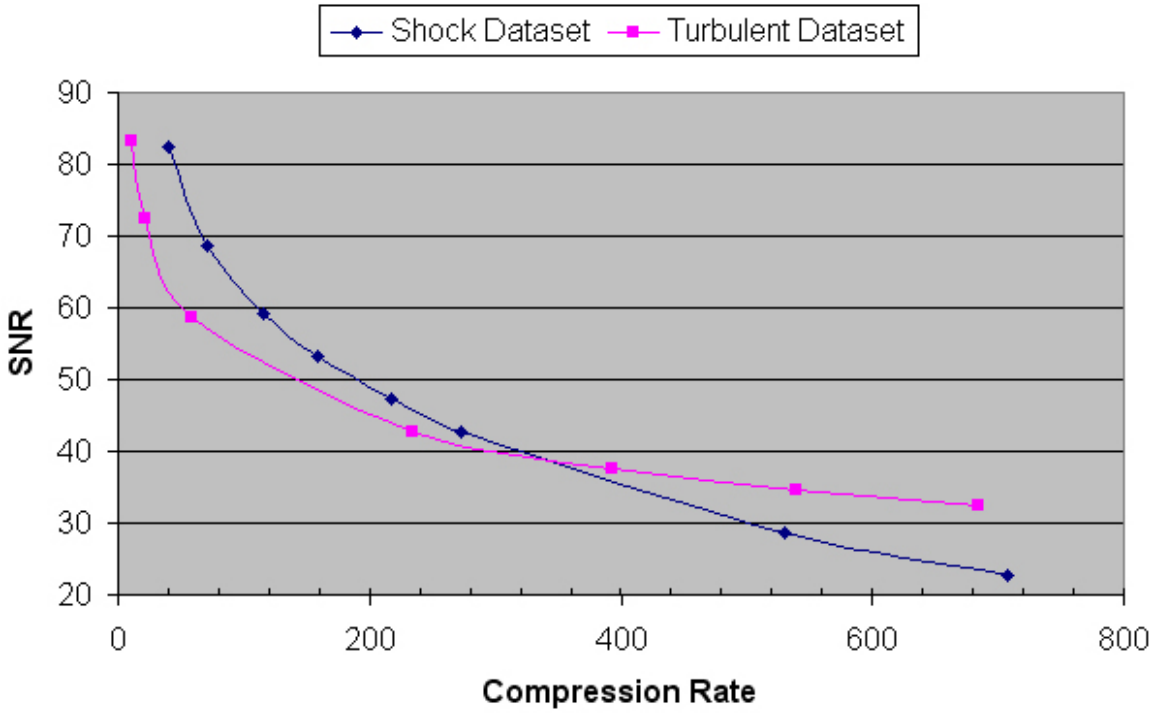


Figure 2.2: SNR for shock dataset and DCT-compressed channel turbulent dataset.

each frame in such cases by interpolating key frames followed by feature reconstruction. A linear interpolation between I-frames is a poor prediction for intermediate P-frames. Instead, we use the underlying physics of the fluid motion to guide the prediction.

The evolution of incompressible fluid with an almost constant temperature can be approximated by solving the following Navier-Stokes equations

$$\nabla \cdot u = 0 \tag{2.7}$$

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \frac{1}{\rho}\nabla p + \nu \nabla^2 u + f \tag{2.8}$$

where u is the flow velocity vector, f an external force, ν a scalar field representing the kinematic viscosity, and ρ the density [3]. Such equations can be approximately solved in real time at low resolutions, such as demonstrated in a smoke motion simulation by Stam [4]. However, due to numerical dissipation and non-linear effects, the numerical error may build up and eventually make the calculation deviate from the real-life solution. To model compressible fluid we would need to consider other terms and equations. When the velocity in compressible fluid is high, small timesteps need to be used to keep the

numerical scheme stable. Such simulation on large grids will likely continue to elude real-time implementation in at least the near future.

We resort to a simple advection rule on a flow field to get an approximation of the flow in the next moment. Many schemes could work. Our choice for simple advection is a semi-Lagrangian algorithm that was first introduced in [4]. We backtrace a point at location x through the velocity $u_t(x)$ over a timestep $\triangle t$ to find a new location $p(x, -\triangle t)$, and use the velocity there as the velocity for x for the next moment:

$$u_{t+1}(x) = \text{Advect}(u_t(x)) = u_t(p(x, -\triangle t)) \tag{2.9}$$

The simple approximation is reasonably close to the actual motion for a few frames.

With the simple advection scheme, a client may use advection onto an I-frame u_{t_i} to approximate following P-frames until the next I-frame $u_{t_{i+1}}$ is reached

$$\tilde{u}_{t_i} = u_{t_i}, \tag{2.10}$$

$$\tilde{u}_{t_i+k} = \text{Advect}(\tilde{u}_{t_i+k-1}) \tag{2.11}$$

The error caused by advection will accumulate. But if the next I-frame $u_{t_{i+1}}$ is not far away, $\tilde{u}_{t_{i+1}}$ will be a reasonable approximation of $u_{t_{i+1}}$.

In the above scheme, however, only u_{t_i} is used to predict P-frames between t_i and t_{i+1}. Actually $u_{t_{i+1}}$ will carry more information about how the frames look like near time t_{i+1}. This observation leads to a so called bi-directional scheme. If we reverse the fluid velocity at $u_{t_{i+1}}$ and start advection, we will reach a state that approximates u_{t_i}.

$$\tilde{v}_{t_{i+1}} = u_{t_{i+1}}, \tag{2.12}$$

$$\tilde{v}_{t_i+k} = \text{ReversedAdvect}(\tilde{v}_{t_i+k+1}) \tag{2.13}$$

In a reversed advection, the flow velocity of the original frame is reversed, then an ordinary advection is applied, then the flow velocity of the resulting frame is reversed again. It is equivalent to advecting the flow backward in time.

$\tilde{u}_t$ and $\tilde{v}_t$ are weighted blended to form the result:

$$\tilde{U}_{t_{i+k}} = \frac{t_{i+1} - t_i - k}{t_{i+1} - t_i} \times u_{t_{i+k}} + \frac{k}{t_{i+1} - t_i} \times v_{t_{i+k}} \tag{2.14}$$

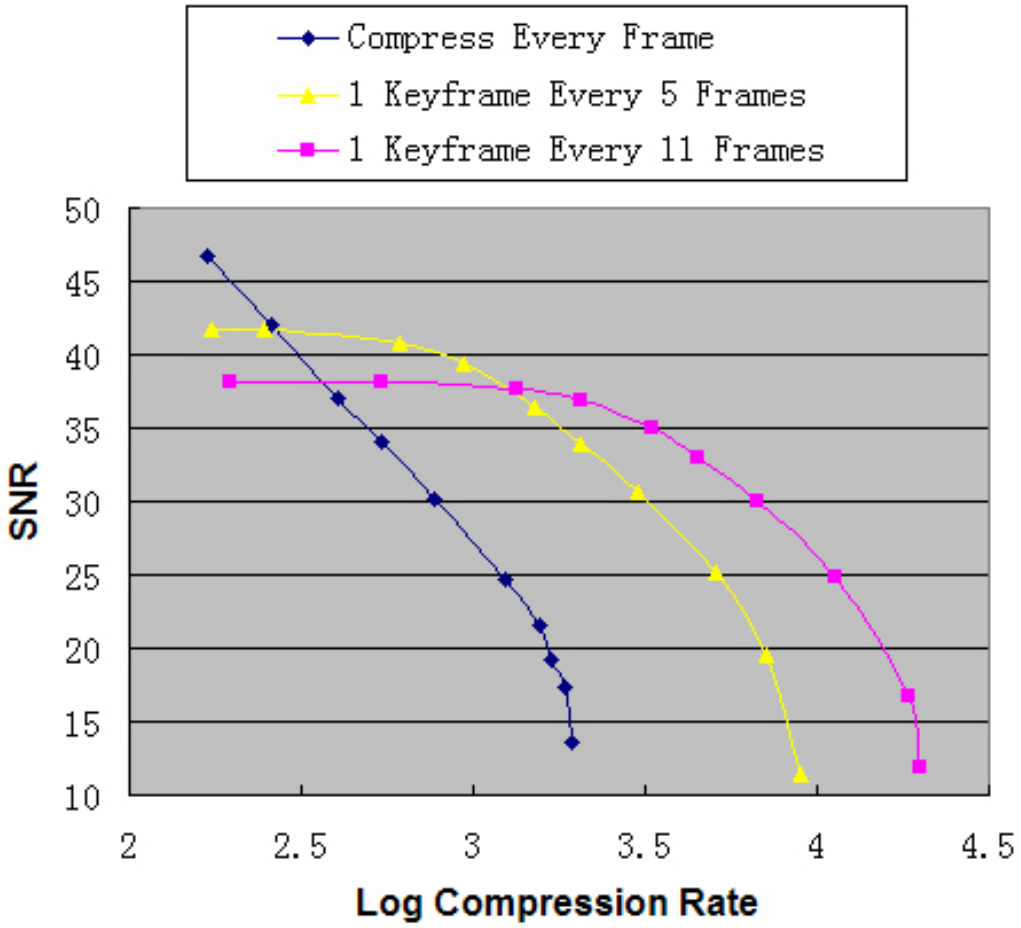


Figure 2.3: SNR for channel turbulent dataset at different key frame periods.

For the applications that put emphasis on providing a quick view of a flow field instead of on reproducing the datasets with a good L^2 norm accuracy, the bi-directional advection scheme offers a much higher compression rate comparing to compressing each frames.

In Figure 2.3, the SNR is compared with different key frame periods for the channel turbulent dataset. When we target at a high compression rate, the bi-directional advection scheme with a larger key frame period gives much better SNR than the per frame compression scheme. On the other hand, advection itself will cause dataset to lose accuracy

between key frames. No matter how well we preserve the key frames at the cost of a lower compression rate, the SNR converges at a lower position than the per frame compression scheme.

Chapter 3

Feature Advection and Reconstruction

The flow datasets decoded from the above method will approximate the original flow motion. However, since the method compresses the flow uniformly regardless of the underlying flow features, it is likely that some important features will be damaged or lost during such process. In this section, we suggest a scheme in addition to the MPEG-like method that allows the client to reconstruct important flow features. Reconstruction methods should be tailored to the specific feature property and to the requirement of the scientists. In this section, we demonstrate the reconstruction scheme of two different features, namely shocks and vortical structures.

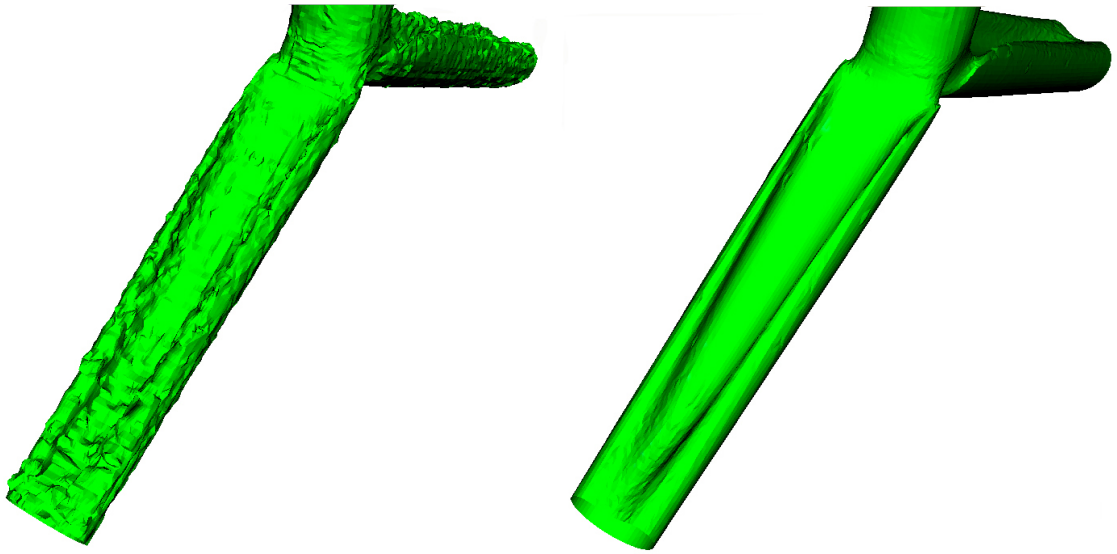
3.1 Shock Reconstruction

A shock surface, here, is defined as the Mach number isosurface in a fluid where discontinuity occurs. It is an important feature for datasets containing supersonic motions. During a bi-directional advection, such an iso-surface might deviate from its original position. To correct the position, we need to modify the recovered dataset.

Shock Structure Compression

Compressing a shock structure requires much less information than compressing the

whole field. Thus it makes sense to send compressed shock structure as Metadata while still obtaining high overall compression rate. For the shear shock in our dataset, flow velocity at both sides of the shock surface have quite parallel directions, but the velocity magnitude changed significantly across the surface. Advection only distorts the correct velocity magnitude, while leaving the flow direction almost unchanged. Such shock structures can be captured by storing the flow vector length at grids on both sides of the shock. During reconstruction, the vector length is applied to the vectors at both sides of the shock to recover the shock. Since shock is a 2D feature in a 3D dataset, such a subset of grids generally only occupies a small portion of the whole grid. The compression scheme in Section 2.4.2 can be applied to this subset of grids. In figure 3.1 we show that such compression is more effective in preserving shock surface than compressing the whole field.



(a) (b)

Figure 3.1: With similar data amount, compressing the whole field in (a) yields a much noisy shock surface than compressing the shock structure alone in (b).

Shock Structure Blending

To correctly reconstruct the shock surface, in addition to blending in the correct surface S, the advected shock S' at wrong position should also be removed. S separates the volume into two S_a and S_b that lie at both sides of the shock, as does S'. Vector fields of the original

dataset at grid layers that are immediately adjacent to both surfaces are compressed as shock Metadata. In Figure 3.2, these vectors are respectively marked as v_a, v_b, v'_a, and v'_b. Advected vector fields on grids within some distance to either of the shock surfaces are to be modified by the shock Metadata.

To blend such a shock structure seamlessly into the advected dataset, different parts of the volume needs to be treated differently. For those grids that lie on different sides of two shocks, (the grey area in Figure 3.2) their advected value is on the wrong side of the actual shock, thus should be abandoned. Their new values come from a distance-weighted interpolation of neighboring shock Metadata that lies on the correct shock side.

For those grids that lie on the same sides of two shocks, (the blue and brown area in Figure 3.2,) the differences between the shock Metadata and the advected data are found along the shock surfaces. Such differences are blended and added back onto vector fields at those grids with a diminishing factor proportional to their distance to the shock surfaces. In this way, the blending results are seamless.

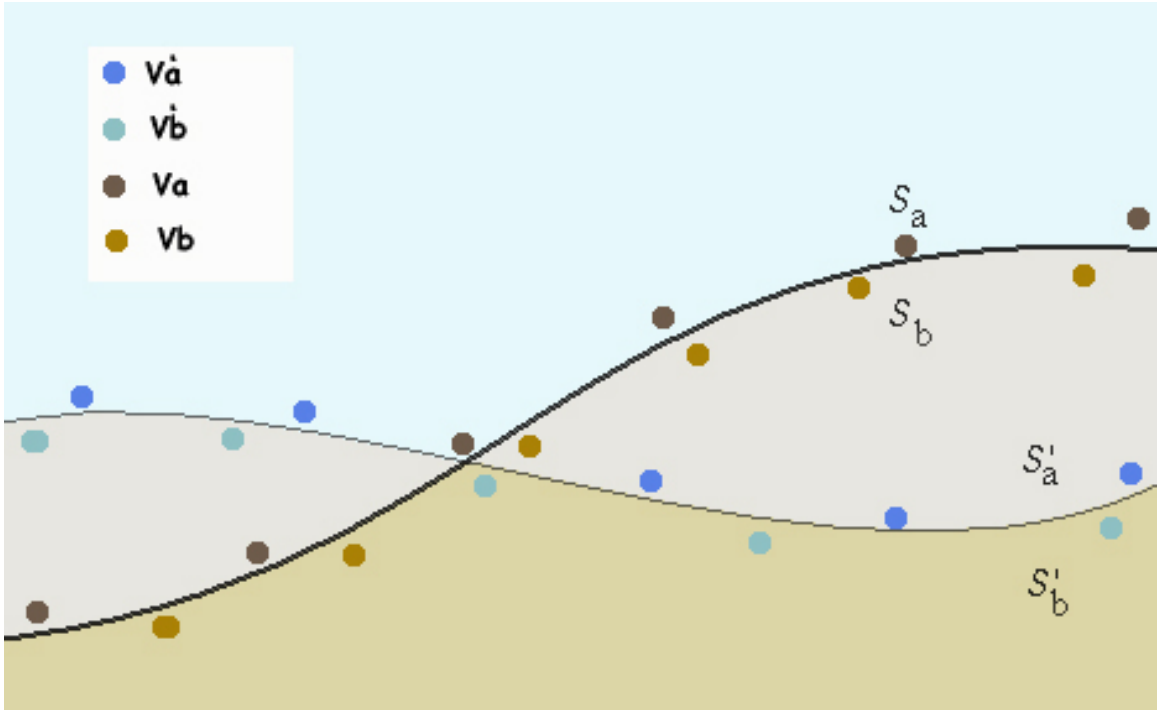


Figure 3.2: Shock S and S' divide the space into three parts. Vectors at both sides of the shock surfaces are interpolated to yield correct shock surface for the client.

3.2 Vortical Structure Reconstruction

It is important to notice that when we advect a flow with a semi-Lagrangian scheme, although the advected velocity field approximates the actual flow field within several frames, many features reflecting first or higher order derivatives of the velocity are much more sensitive to the inaccuracy introduced by the advection scheme. In Figure 3.8, vorticity iso-surface is found for a flow dataset. In Figure 3.9, vorticity iso-surface is lost during advection.

If we do a feature advection by finding flow features for each I-frame and applying bi-directional advection onto them, we may approximate how those features actually evolve in time. Then we may generate a field W_t with desired features F_t for each timestep and blend it into flow data.

With the above observation, our feature reconstruction scheme adds the following steps in FlowPEG. Assume that an important feature F should be preserved. After advection, F_t is distorted as $\tilde{F}_t$ in $\tilde{u}_t$ at time t. First, feature detection is applied at each I-frames to yield F_i. Bi-directional advection is applied to find F_{i+k} at all P-frames. A field W_t is constructed at each timestep that contains F_t.

A feature detection is also applied to all P-frames to detect $\tilde{F}_t$. A field $\tilde{W}_t$ is constructed that contains $\tilde{F}_t$. The final flow data can be constructed as

$$U_t = \tilde{U}_t - \tilde{W}_t + W_t \tag{3.1}$$

A quick construction of W_t for any user specified feature F_t is crucial for such a scheme. We apply this scheme on an important flow feature, the flow vorticity.

Reconstructing vortical structures

Vortical structures are vital in the research of chaotic fluids. They are defined by the curl field of a given vector field U.

$$\omega = \nabla \times U \tag{3.2}$$

However, such structures are severely damaged during the advection scheme. We need to remove damaged vortical structures from the resulting flow and add back bi-directionally advected vortical structures from I-frames. These two steps both require to construct a pure rotational vector field W from a given curl field ω.

$$\nabla \times W = \omega \tag{3.3}$$

$$\nabla \cdot W = 0 \tag{3.4}$$

It can be shown that W can be found by solving three independent Poisson equations

$$\nabla^2 W_x = -(\nabla \times \omega)_x \tag{3.5}$$

$$\nabla^2 W_y = -(\nabla \times \omega)_y \tag{3.6}$$

$$\nabla^2 W_z = -(\nabla \times \omega)_z \tag{3.7}$$

A linear solver may be used to solve this problem. However, it worth noting that ω is the curl field of some vector field only when $\nabla \cdot \omega = 0$. When we apply bi-directional advection on ω_i at I-frames, the interpolated $\tilde{\omega}_t$ is generally not divergence free. In that case, the solver will converge at a solution for $\nabla \times W = \omega'$, where $\nabla \times \omega' = \nabla \times \omega$ and $\nabla \cdot \omega' = 0$. ω' removes the divergence part of ω, thus is a better approximation of intermediate vortical structures between I-frames. In Fig. 3.10, the vortical structures are reconstructed during the visualization.

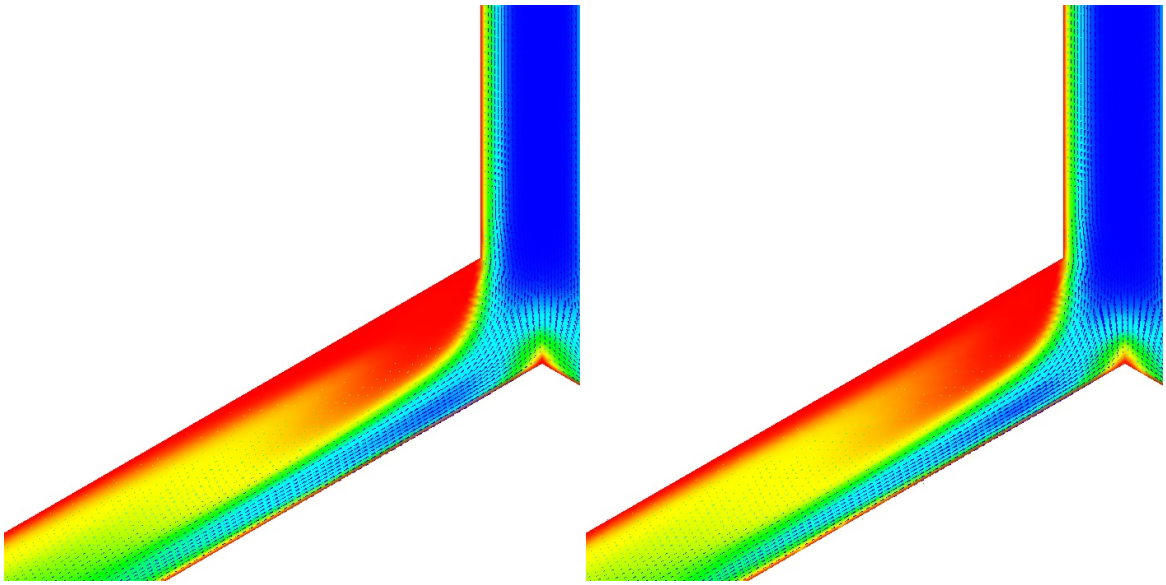
Due to the Semi-Lagrangian advection and the smoothing caused by blending forward and backward advected data, the advected vortical structure also suffers some dissipation. It is compensated by a rescaling on the vortical magnitude according to the spectrum of nearby key frames. Such reconstruction is restricted within the volume to where interesting vortical structure presents. The resulting field is blended into the rest of the dataset.

3.3 Results

3.3.1 Dataset with Shock Structure

The shock dataset represents 76 time varying solutions for non-Newtonian flow in a y shaped pipe. The grid is unstructured mixed element containing hexahedra, tetrahedra, prisms and pyramids. The fluid flows in from one of the pipes, and the shock formed to propagate into the other two pipes starting at the 20th frame of the simulation. During the key frames compression, the space is subdivided into uniform cubic regions. The largest grid number within a region is 610, with an average of 244. Fifty percent (or 50, whichever is smaller) of the eigenvectors with lower frequency are used in compression, while the rest high frequency eigenvectors are assumed as visually insignificant, and are therefore excluded from compression. Key frames are selected every 15 frames. After bi-directional advection is applied to interpolate frames in between, the shock surfaces deviated from its actual location. The original shock is located between two layers of grid, and the deviation of the advected shock is measured by the number of grid layers from the correct location. Such deviation is captured with the color coding in Fig. 3.5. During shock reconstruction, the shock surface at two lower pipes are regarded as interesting features. After shock reconstruction, the shock surface appears slightly noisy due to lossy compression of the shock structure. But the recovered shock surface is confined to correct grid layers in Fig. 3.6. In Fig. 3.3 a slice of cutting plane shows the similarity of the flow field itself.

FlowPEG compressed the dataset to 0.33% of its original size with bi-directional advection. After adding the Metadata for shock structure in the lower pipes, the compressed size is increased to 0.88%, with shock surface confined within correct layers of grids. The average decompression time for each frame on a Xeon 2.66G CPU is 0.31 second, compression time on the server side adds 0.27 second. The bi-directional advection from key frames takes 4.0 seconds per frame, which is mostly kd-tree search on unstructured points. The compression of shock structure for each frame involves eigen vector finding for hun-



(original) (recovered)

Figure 3.3: Cutting planes comparing the original shock flow with the flow reconstructed.

dreds of points in divided regions. Shock correction takes 20.0 seconds on the server, and 19.2 seconds on the client, with 14.1 seconds used by switching to Matlab to find eigenvectors for the shock structure. Eigenvector finding speed also depends on the size of the actual shock surface. For the shock surface occupying most of the two lower pipes, the time reaches 47 seconds.

3.3.2 Dataset with Vortical Structure

The resolved vorticity field which is used as a reference field is generated from the turbulent channel flow simulation at a fairly low Reynolds number. The simulation is performed by solving the time dependent compressible Navier-Stokes equations governing the conservation of mass, momentum and energy. The domain size is $2\pi h \times 2h \times 4/3\pi h$ in the streamwise, wall normal and spanwise direction, respectively, and represented using grid resolution of $64 \times 128 \times 64$ in the corresponding directions. The symbol h denotes the half channel height. The boundary conditions used are no-slip condition for the planar walls

and periodic condition in the wall parallel directions. This channel flow is a standard case for wall bounded turbulence validation. The flow is initialized with the laminar parabolic profile of the streamwise velocity and constant value for the other flow variables (pressure, density, and temperature), representing the normal room condition. Velocity perturbations are added to the laminar base flow to trigger instabilities and ensure the regularity of the vorticity field. The perturbations are sinusoidal in the streamwise and spanwise direction. We employ a second order central finite volume method for the spatial discretization and explicit four stage Runge-Kutta for the time stepping. The time step of the resolved simulation is about $2.5E-07$ seconds and the solution frame is stored every $1.E-5$ seconds (40 time steps).

The flow is transitioning from the laminar to turbulent state through non-linear interaction of the disturbances. As can be seen in the time development of the vorticity field, the regular and simple vorticity structures at the start condition break down to increasingly smaller and irregular structures with time due to the non-linear interaction. The peak of the vorticity intensity is located near the walls, whereas the minimum intensity at the center plane of the channel. The solution presented in the current study represents the start of the transition process, therefore the footage of the regular pattern is still clearly present. In a later stage of transition or fully turbulent stage the structures are completely chaotic. The statistics of a fully turbulent channel flow however possess some characteristic properties. For instance the mean flow profile, power spectrum, and Reynolds stresses of the flow have distinct shapes governed by specific relations or formulas. These properties can be used as a quality measure of the reconstruction algorithm built in a flow visualisation program that attempts to preserve certain flow features.

During the compression of this dataset, one I-frame is selected every 10 frames. Then 3D DCT is applied to each I-frame, which in turn is sent to the client. The overall compression rate is 1019.7 for 100 frames. The client applies bi-directional advection to interpolate on P-frames. Feature advection is also applied on the curl fields. Vortical structures are

reconstructed from them and replace the rotational part of advected flow. The curl field is sampled at half of the resolution for faster reconstruction.

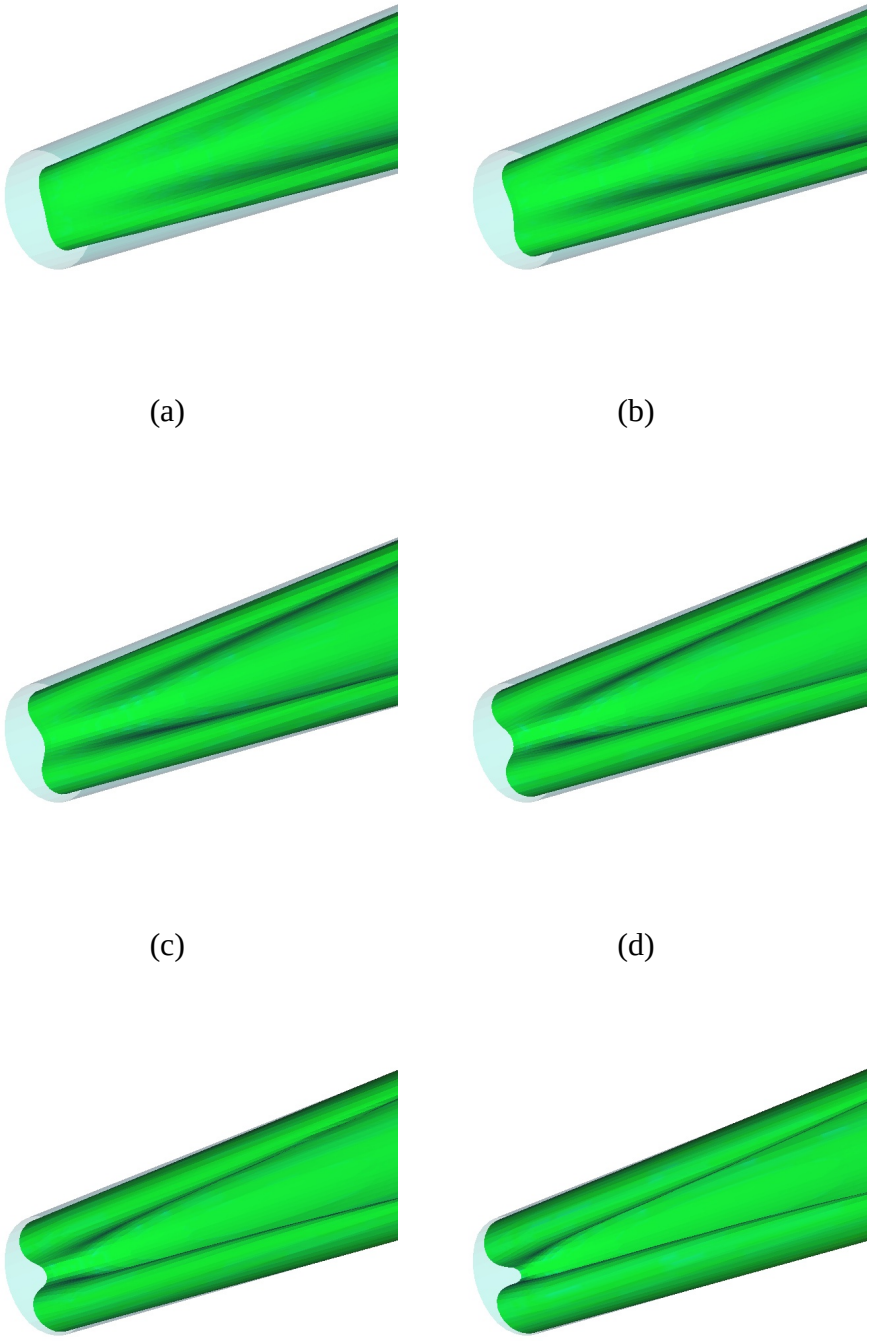
The typical running time of this algorithm is measured on a Xeon 2.66G CPU. The average decompression time for each frame is 0.24 second, compression time on the server side adds 0.23 second. Bi-directional advection takes 1.26 second per frame. Vortical structure reconstruction takes 6.25 seconds.

A cutting plane shows flow velocity for both original data and recovered data in Fig. 3.7. They demonstrate similar flow patterns. An iso-surface of the original vorticity is shown in Fig. 3.8. After bi-directional advection, the iso-surface remains at I-frames in (a) and (f), but it is lost during intermediate P-frames in (b), (c), (d) and (e). The structure is recovered by encoding the curl as a separate data channel. The vorticity becomes weak during the P-frames because it suffers some dissipation during feature advection, but maintains overall shapes.

3.4 Conclusion

In this paper we propose a new compression scheme, called FlowPEG, designed to transmit compressed time-varying vector flow datasets. In this scheme, the decompression on the client uses inexpensive bi-directional advection to approximate the flow behavior. Important features like shock and vortical structure are reconstructed on the client. With this algorithm, a moderately configured personal computer can receive and visualize otherwise very large datasets stored on a remote server.

Our technique employs existing scalar field DCT methods for compressing static vector field frames on a regular grid, and uses a novel Laplacian eigenvector based compression scheme for data on unstructured points. As has been demonstrated in the paper, FlowPEG appears promising and better allocates compression resources to interesting area of the flow.

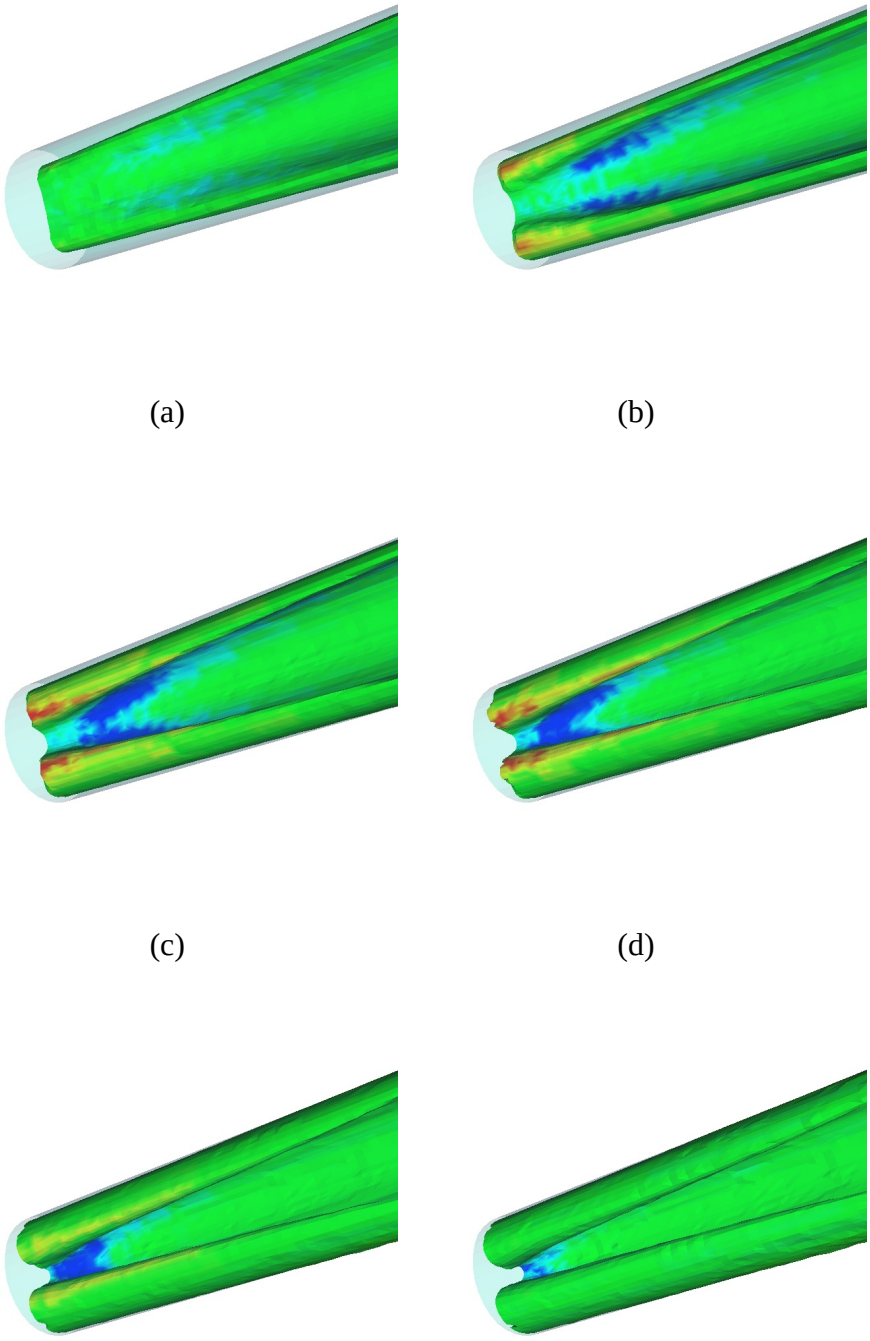


(a) (b)

(c) (d)

(e) (f)

Figure 3.4: The original shock surfaces.

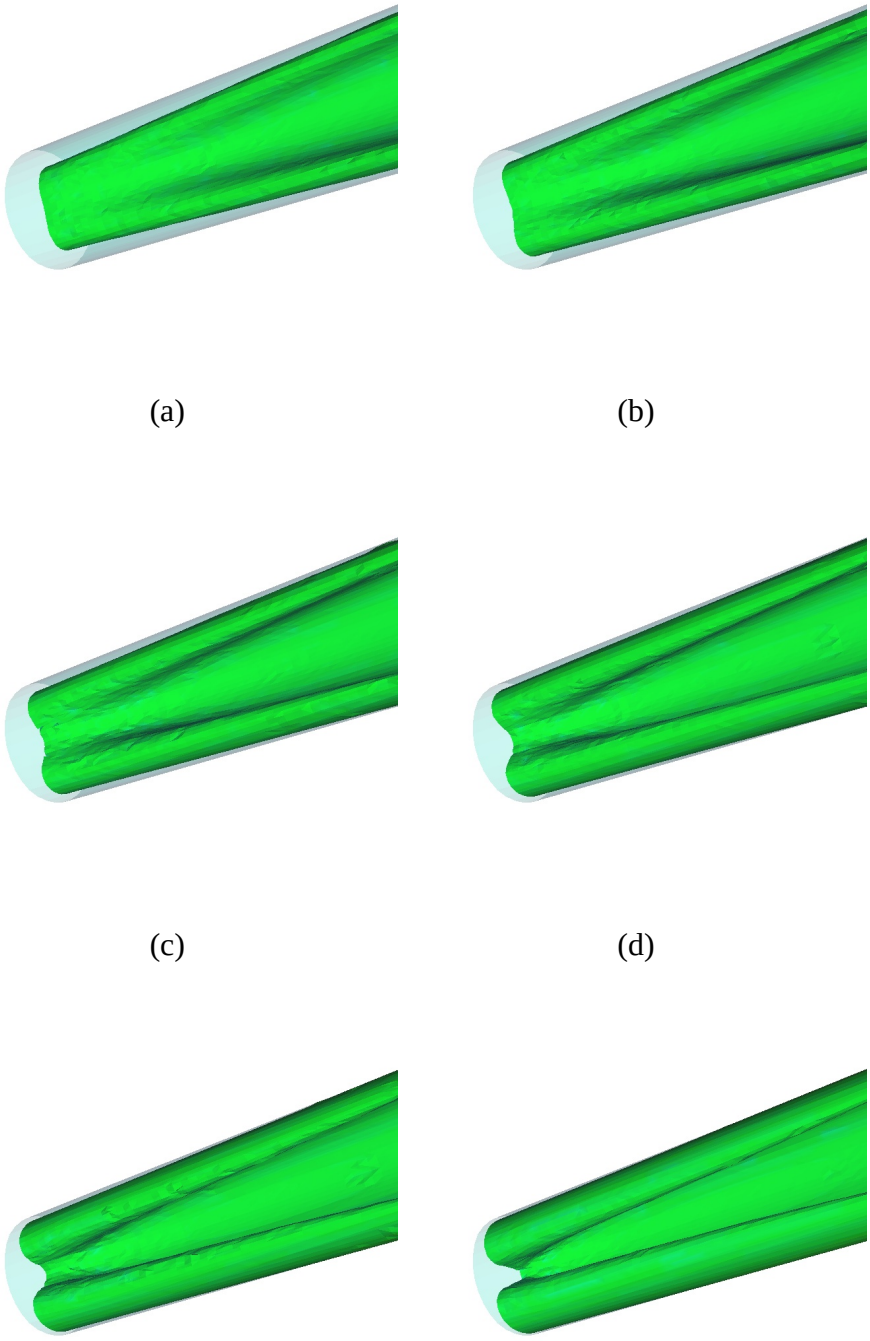


(a) (b)

(c) (d)

(e) (f)

Figure 3.5: The advected shock surfaces.

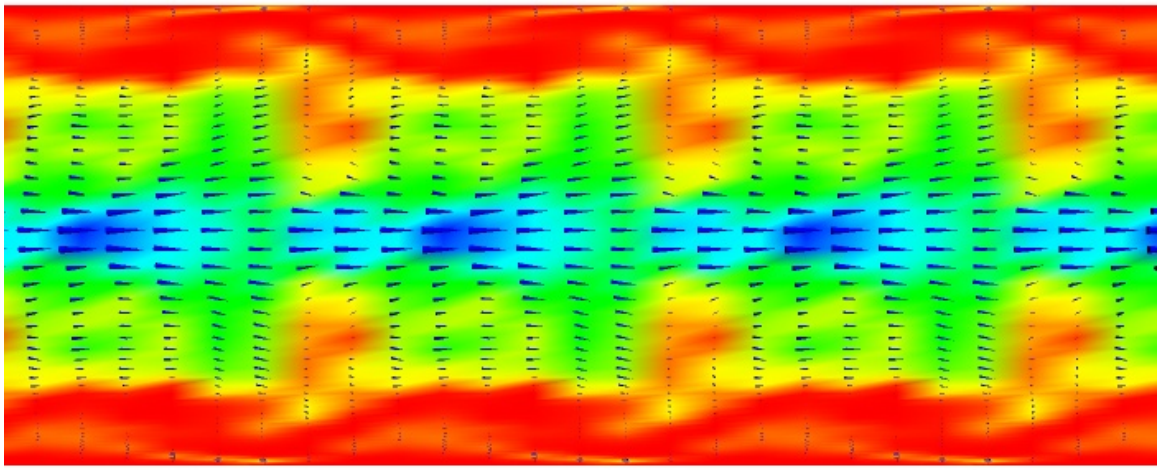


(a) (b)

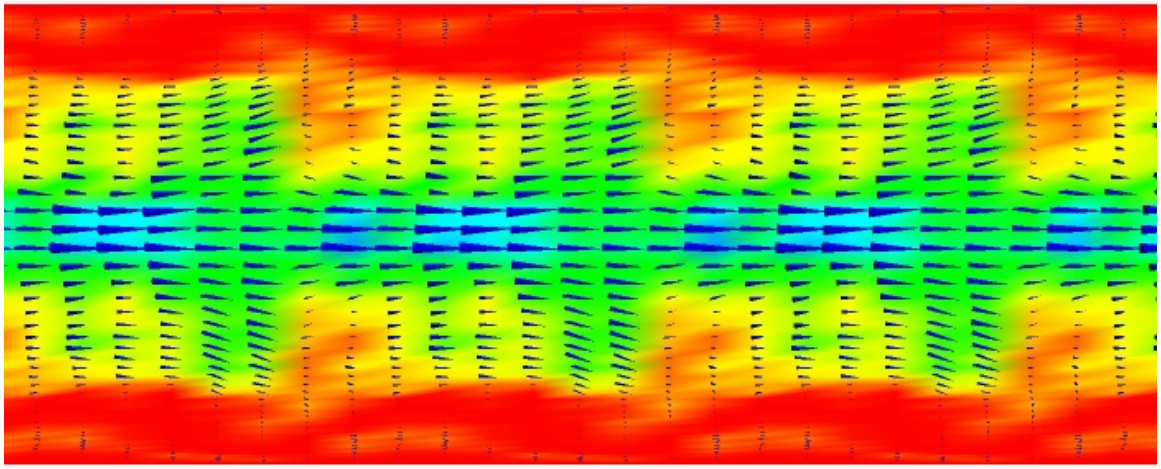
(c) (d)

(e) (f)

Figure 3.6: After reconstruction the surfaces are confined within correct layers of the grid.

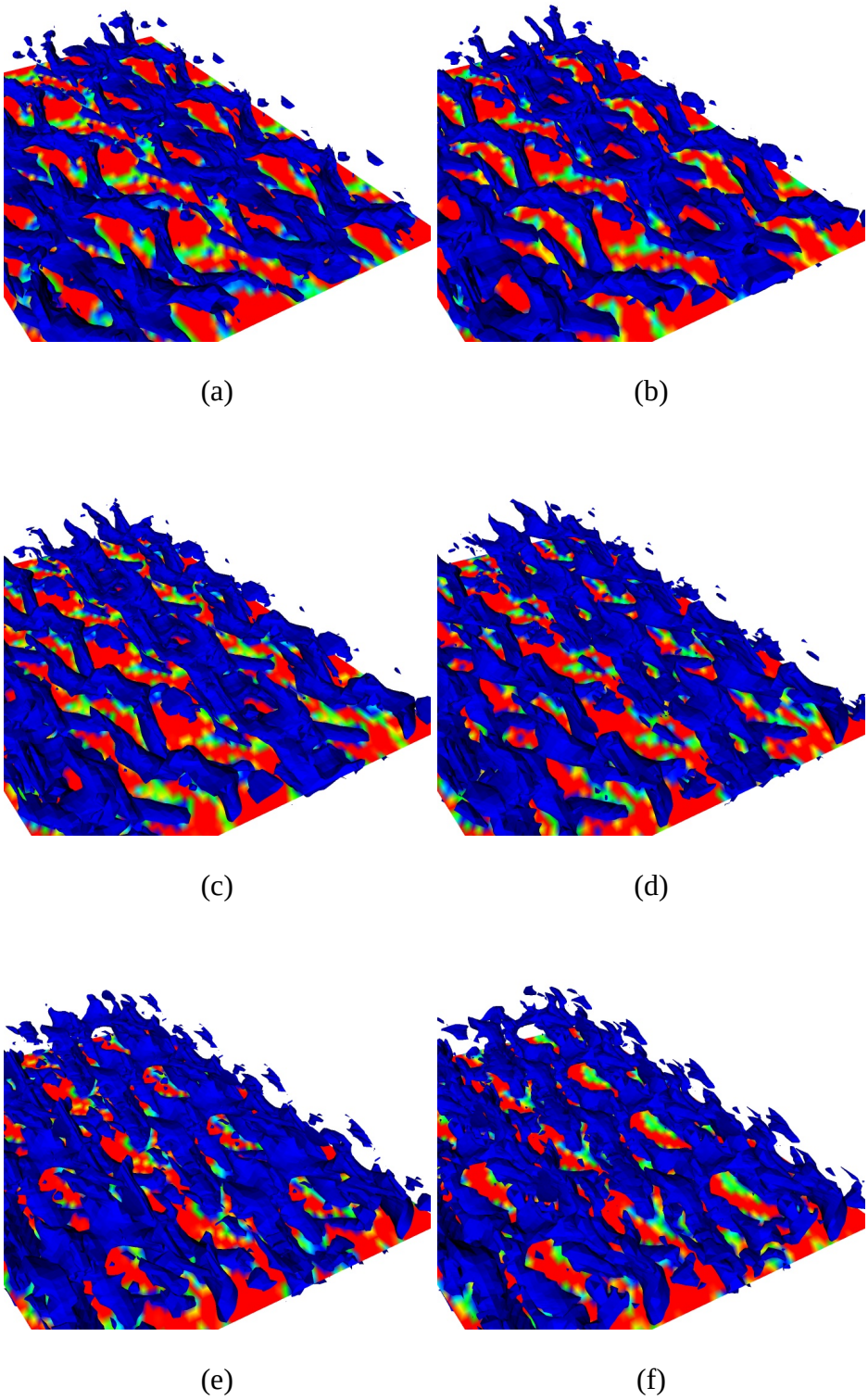


(original)



(recovered)

Figure 3.7: Cutting planes comparing the original flow with the flow reconstructed from the compressed representation.

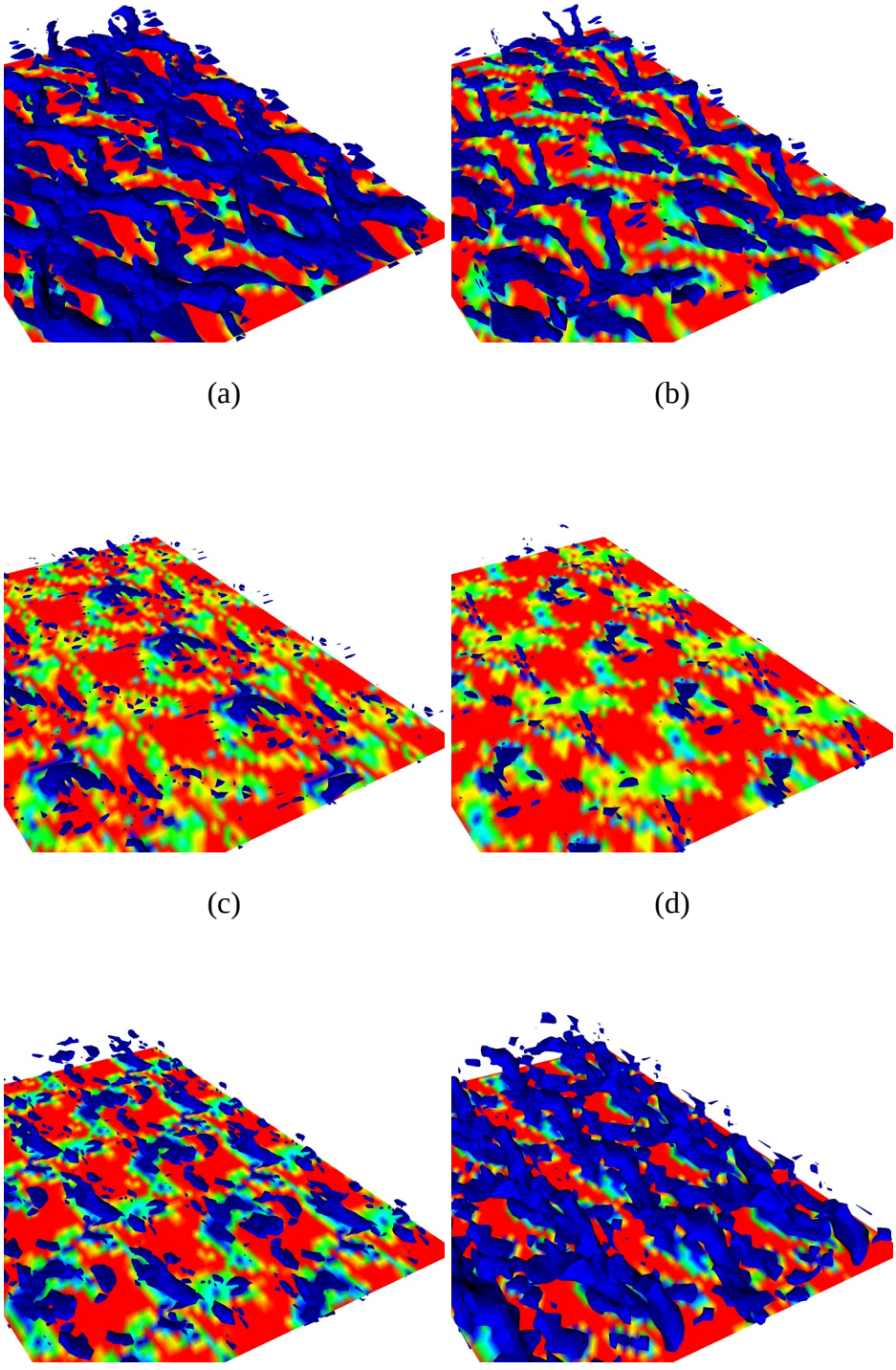


(a) (b)

(c) (d)

(e) (f)

Figure 3.8: An iso-surface rendering of the curl of the original flow.

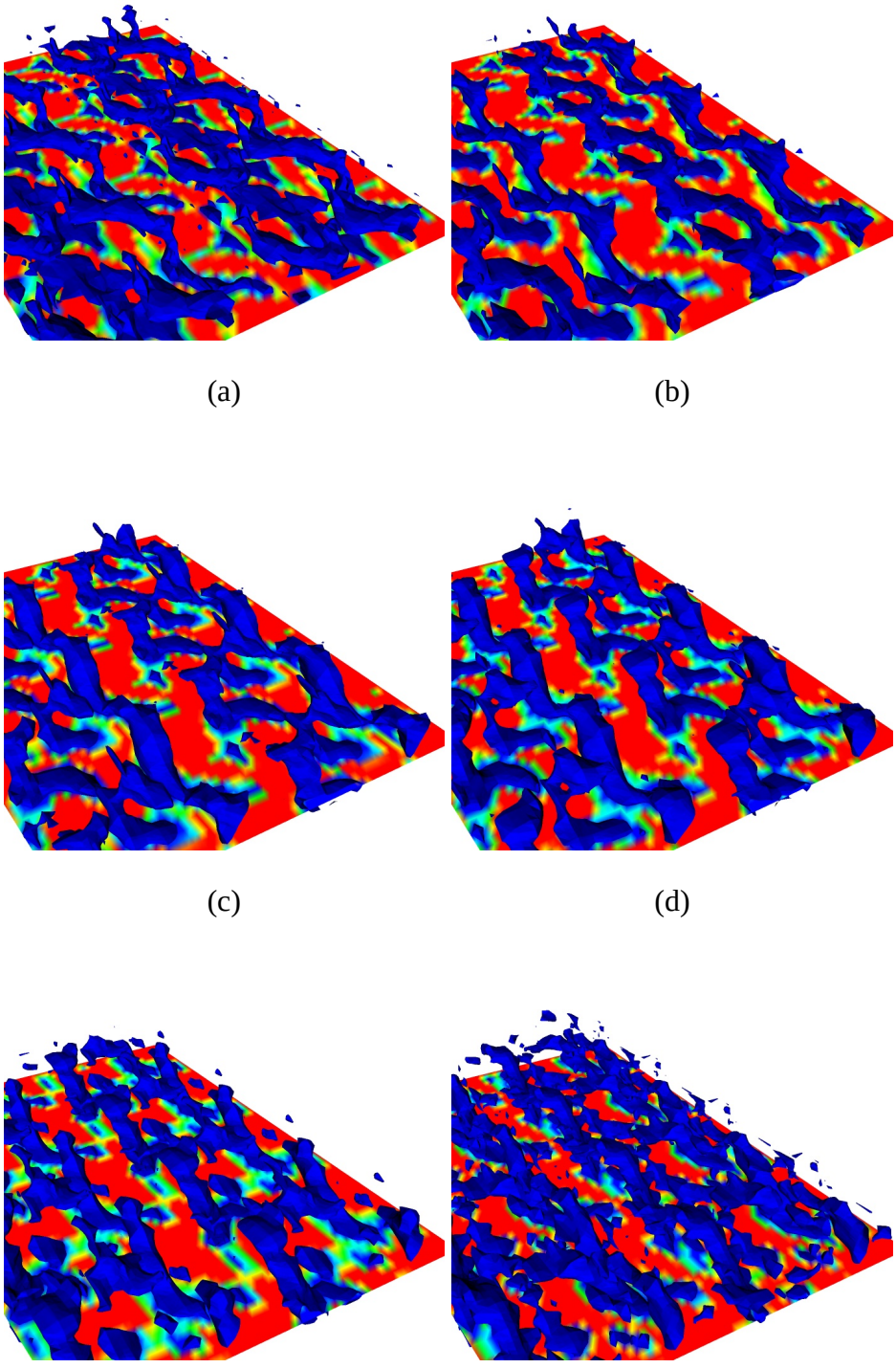


(a) (b)

(c) (d)

(e) (f)

Figure 3.9: The advection used to interpolate between I frames at each end destroys the rotational components of the flow.



(a) (b)

(c) (d)

(e) (f)

Figure 3.10: Extracting and encoding a separate curl field enables FlowPEG to accurately reconstruct the rotational components of the compressed flow data

Chapter 4

Textureshop

Textureshop is a tool that combines shape from shading and distorted graphcut texture synthesis to conveniently and robustly allows a user to texture an object in a photograph. We create small pixel patches on the surface in a photo, clustered by similar recovered normals. Texture synthesis are performed on these patches, distorting pixel positions into a local parameterization to account for patch orientation and displacement. These patches are further distorted to match the features of neighboring patches. This patchwork of feature-aligned foreshortened textured pixel clusters gives the illusion that the texture is applied to the photographed surface. We furthermore apply shape-from-shading to the texture to support displacement mapping and normal transfer (embossing).

4.1 Shape from Shading

A wide variety of sophisticated shape-from-shading and photoclinometric algorithms exist for reconstructing a surface from an image [24]. These methods pose shape-from-shading as an optimization problem and employ iterative methods to solve the resulting partial differential equations. The results are gradient and height fields consistent with the surface portrayed in the image.

For visually plausible texture synthesis, we need not find a strictly consistent height

field nor construct a full surface representation. We will instead segment the surface into oriented patches, and texture synthesis can be performed independently upon these patches.

4.1.1 Normal Recovery

Horn [24] gives the formulae for recovering the surface normals from an image for a wide variety of reflectance functions, but we find the following simple Lambertian reflectance model works well for our purposes. Let S be the unit vector from the center of the object toward a sufficiently distant point light source. We further assume the point on the surface with largest intensity I_{max} faces the light source. The darkest point is shadowed and its intensity I_{min} indicates the ambient light in the scene. The function $c(x,y) = (I(x,y) - I_{\min})/(I_{\max} - I_{\min})$ estimates the cosine of the angle of incidence, and $s(x,y) = \sqrt{1-c(x,y)^2}$ its sine, which leads to the recovered normal $N(x,y)$ as

$$G(x,y) = \nabla I(x,y) - (\nabla I(x,y) \cdot S)S, \tag{4.1}$$

$$N(x,y) = c(x,y)\, S + s(x,y)\, G(x,y)/||G(x,y)|| \tag{4.2}$$

where $\nabla I(x,y) = (\partial I/\partial x, \partial I/\partial y, 0)$ is the image gradient.

We estimate the vector to the light S from the intensity of pixels (x_i, y_i) on the boundary of the object's projection. For such pixels the normal $N(x_i, y_i)$ is in the direction of the strong edge gradient. The source vector S is then the least-squares solution to the overconstrained linear system

$$N(x_i, y_i) \cdot S = \frac{I(x_i, y_i) - I_{\min}}{I_{\max} - I_{\min}}. \tag{4.3}$$

The user can adjust the light source direction manually if the inferred result is incorrect.

Such an estimated normal field is generally inaccurate, but it captures the undulations of a surface well enough to support texture synthesis. It is also very fast, thus suitable for an interactive photograph editing tool.

4.1.2 Interactive Normal Editing

The normal field recovered from the brightness of an image can be unsatisfying due to various reasons: multiple light sources, non-Lambertian materials and textured materials. Moreover, shading information is lost in shadowed areas. Rather than attempting to handle these complications automatically, we instead implemented several intuitive methods for tuning the result interactively.

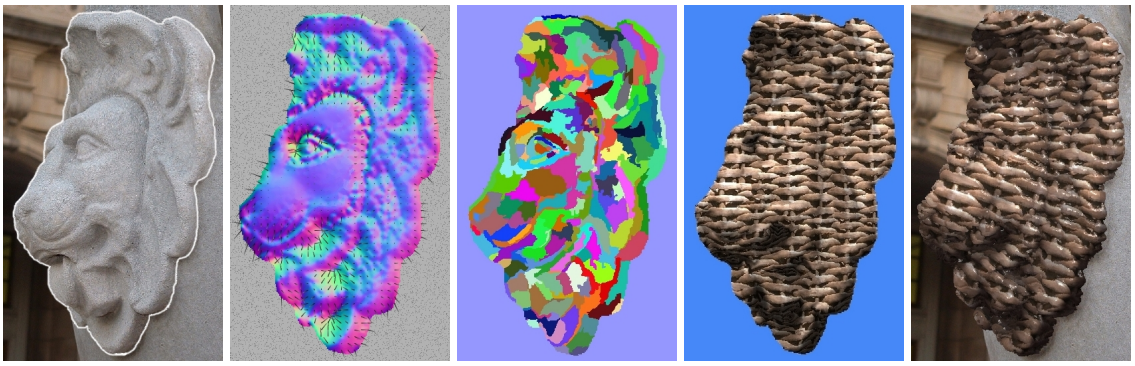
We display the normal field as an RGB-coded normal image, a vector field, or with a quickly synthesized texture. The user can manipulate the reflectance model of the surface with a spline curve initialized to the Lambertian reflection model. The influence of surface texture and noise can be reduced by smoothing the image intensity and/or the recovered normal field. The user can also rotate selected normals to compensate for the effect of a second light source. Finally, the variation of normals over a region can be manipulated, as demonstrated in Fig. 4.2.

4.1.3 Surface segmentation

The surface pixels are grouped into patches with similar normal directions using a bottom-up scheme. The segmentation process is initialized by assigning each pixel to its own patch. For each patch P_i, let N_i, C_i and $|P_i|$ denote the patch's mean normal, centroid pixel and number of pixels, respectively. Then two neighboring patches P_1, P_2 are merged if the error metric

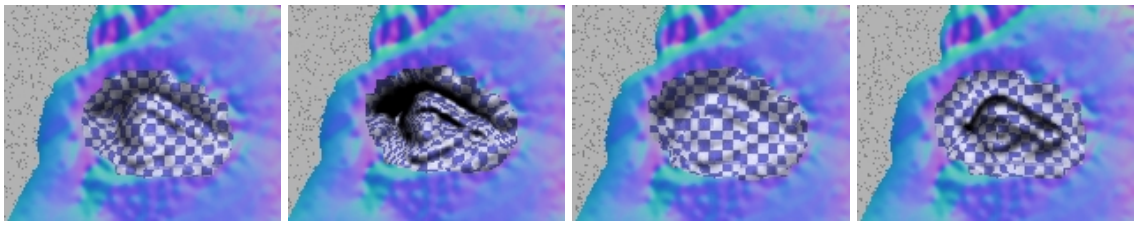
$$E(P_1,P_2) = k_1\sqrt{1-N_1\cdot N_2} + k_2||C_1-C_2|| + k_3(|P_1|+|P_2|) \tag{4.4}$$

falls below a given threshold. Appropriate settings for the constants $k_{1,2,3}$ will yield moderate-sized round patches of similarly oriented pixels. In most cases we used $k_1 = 187$, $k_2 = 20$, $k_3 = 1$ except for Fig. 4.1(c) which used a larger k_1 to further emphasize orientation clustering. The patches are then expanded by a fixed-width boundary (8 pixels in our examples) so they overlap each other.



(a) (b) (c) (d) (e)

Figure 4.1: (a) An ordinary photo is scanned in. (b) Shape-from-shading applied to an object in the image to estimate its normals. (c) Pixel patches formed by clustering normals. (d) Texture synthesized on these patches and aligned with neighboring patches. (e) Final result with texture orientation distortion, displacement mapping and environment mapping.



(a) (b) (c) (d)

Figure 4.2: Variation of the normal field (a) is enhanced (b), reduced (c) and reversed (d).

4.2 Patch Distortion

Though the patches have been formed, simply applying the graphcut texturing algorithm [25] on them will yield a flat texture. We describe several patch distortions that result in a more realistic surface texturing appearance.

4.2.1 Patch Orientation

To achieve the illusion that the texture follows the underlying surface, a patch orientation distortion algorithm will assign each pixel $P(x,y)$ a new position in its texture coordinates $U(x,y) = (u,v)$ to capture the foreshortening distortion due to its recovered normal.

The algorithm starts at the center pixel $P(0,0)$ of the patch, setting its texture coordinates $U(0,0) = (0,0)$, and propagates the distortion to the rest of the patch in a width-first floodfill order.

Let $P(x,y)$ indicate the pixel at (x,y) with distorted position $U(x,y)$ and recovered (unitized) normal $N(x,y) = (N_x, N_y, N_z)$. Given $P(x,y)$ we compute the foreshortening distortion of the next pixel to its right $P(x+1,y)$ by projecting this pixel's position $(x+1,y,0)$ onto the recovered tangent plane of pixel $P(x,y)$ and then rotating this projection back into the image plane, as illustrated in Fig. 4.3. The distortion is cumulative and propagates by adding the resulting offset to the current distortion $U(x,y)$ and storing the result in $U(x+1,y)$.

The projection of the point $(x+1,y,0)$ onto the plane with normal $N(x,y)$ passing through $(x,y,0)$ is $(x+1,y,-N_x/N_z)$. Let θ be the angle between N and $Z = (0,0,1)$ and abbreviate $c = \cos\theta = N_z$ and $s = \sin\theta = \sqrt{N_x^2 + N_y^2}$. The unitized axis of rotation is $(N \times Z)/||N \times Z|| = (N_y/s, -N_x/s, 0)$ which leads to the rotation matrix

$$R = \begin{bmatrix} c + (1-c)N_y^2/s^2 & -(1-c)N_xN_y/s^2 & -N_x \\ -(1-c)N_xN_y/s^2 & c + (1-c)N_x^2/s^2 & -N_y \\ N_x & N_y & N_z \end{bmatrix}. \tag{4.5}$$

The product $R(1,0,-N_x/N_z)$ yields the new position of pixel $P(x+1,y)$, leading to the

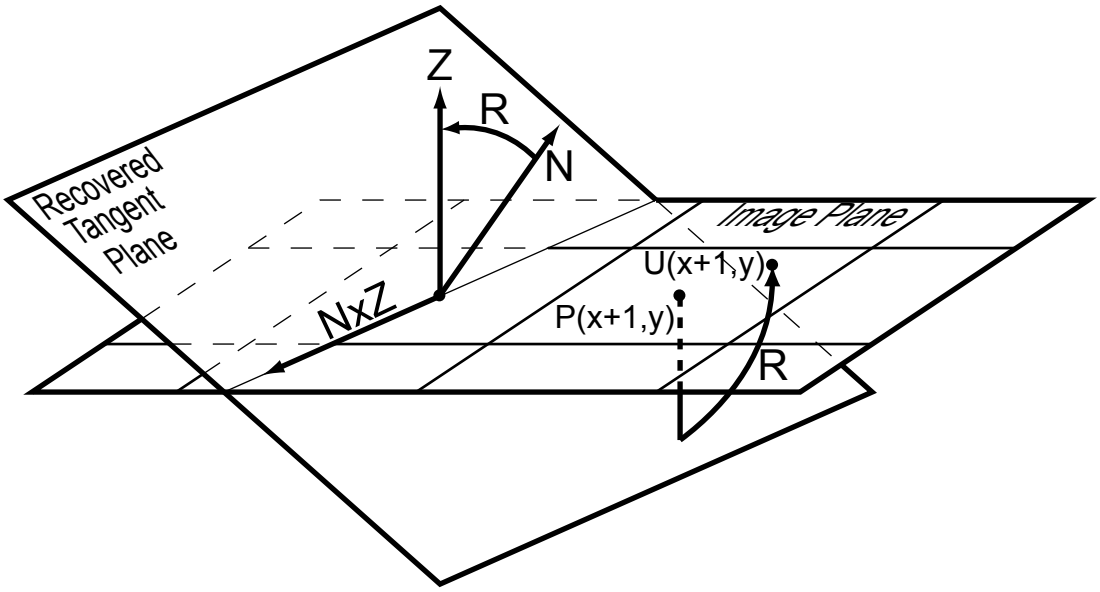


Figure 4.3: Texture distortion according to recovered patch orientation.

propagation rules

$$U(x \pm 1, y) = U(x,y) \pm \frac{(1 + N_z - N_y^2, N_x N_y)}{(1 + N_z) N_z}, \tag{4.6}$$

$$U(x, y \pm 1) = U(x,y) \pm \frac{(N_x N_y, 1 + N_z - N_x^2)}{(1 + N_z) N_z}. \tag{4.7}$$

We clamp N_z to a minimum of 0.1 and renormalize N_x, N_y to avoid outrageous distortions. If the distortions of more than one neighboring pixel are available for propagation, then the final orientation distortion is the mean of the distortions computed from each of these neighbors. This averaging reveals that this scheme generates an inconsistent parameterization, and these inconsistencies increase in severity with distance from the centroid, but our segmentation heuristic is designed to produce small round patches that reduce the variance of their normals to keep these internal inconsistencies small, and the inconsistencies between patches are later camouflaged by the feature-sensitive seams cut through overlapping areas by the graphcut algorithm.

4.2.2 Texture Orientation

Texture orientation can also be defined over the image, to more consistently align anisotropic features of the synthesized texture. Vector field orientation can be modified by dragging the mouse over the photo. The patch parameterization is then rotated about its centroid (conveniently the origin of the parameterization) to align the preferred texture direction vector with the appropriate axis of the texture swatch. The rotation of the parameterization effectively rotates the patch about its average normal.

4.2.3 Displacement Mapping

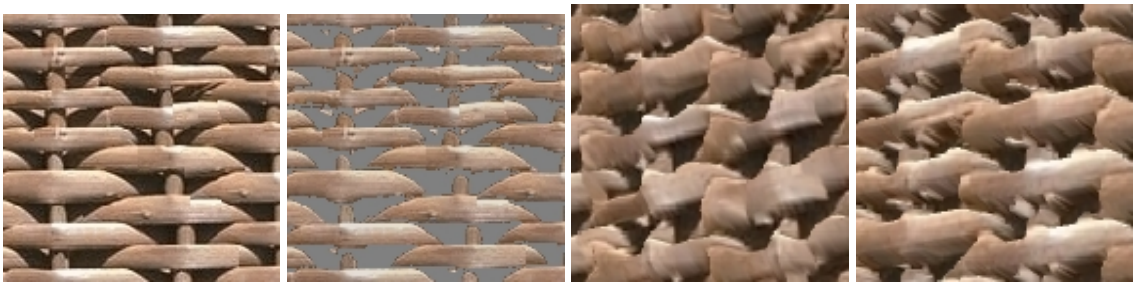
We have thus far applied shape-from-shading techniques to the photograph to recover a local surface on which to synthesize a texture. We can also apply shape-from-shading to the texture swatch used as a source for texture synthesis. This will allow us to do displacement mapping on surface.

We predict the normals $\widehat{N}(x,y)$ of the texture swatch using the same method from the previous section. But whereas the photographed object surface was reconstructed locally, the texture swatch will require a global surface reconstruction. Assuming the input texture color variation is caused only by local normal changes, the height field of the texture swatch $h(x,y)$ is determined by the Poisson equation

$$\nabla^2 h(x,y) = \nabla \cdot \widehat{N}(x,y) \tag{4.8}$$

and solved by conjugate gradients. The user-specified height of a portion of the texture serves as a boundary condition (e.g. the shadowed area of Fig. 4.4(b) was assigned a height of zero).

Often features reconstructed by (4.8) will shrink or grow when compared to the original. In Fig. 4.4(c), the reconstructed wicker is too narrow, but can be interactively corrected by a user-specified nonlinear scale of the height field, yielding Fig. 4.4(d).



(a) (b) (c) (d)

Figure 4.4: Basket texture original (a). Height of shadowed area is set to zero (b). Wicker of the basket is narrow in recovered result (c). Wicker become wider after a non-linear scaling on the height field (d).

During displacement-map texture synthesis, each texture sample is translated in the direction of the photograph's image-projected recovered normal $(N_x, N_y, 0)$ by the recovered texture height $h(x,y)$ foreshortened by the recovered texture normal $\sqrt{1-\widehat{N}_z^2}$. We upsample both the surface normal and texture height to avoid holes. An example is shown in Fig. 4.5 (b).

(a) (b) (c)

Figure 4.5: Texture synthesis from a source image texture without displacement mapping is betrayed by its smooth silhouette (a). Applying shape-from-shading to the source texture produces a noisy displacement mapping (b) that is fixed by upsampling and filtering (c).

These displacements are significant enough to cause aliases when a texture, such as wicker, contains sharp edges. These artifacts can be sufficiently reduced by blending the edge samples with Painter's algorithm according to the percentage of the pixel they cover, as shown in Fig. 4.5(c).

4.2.4 Feature Matching

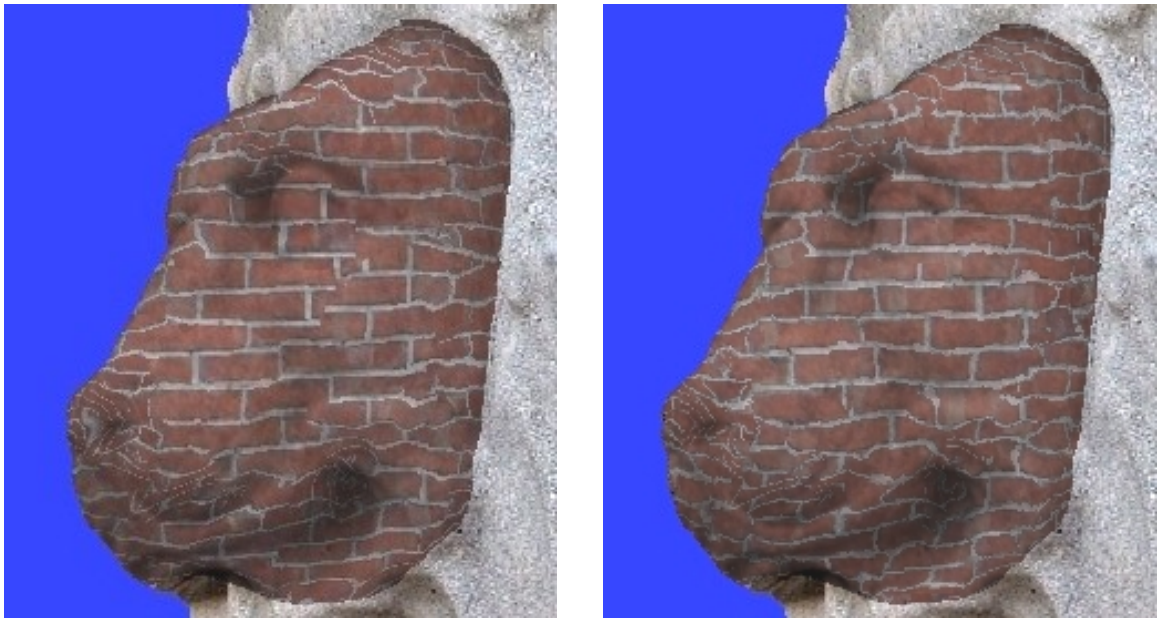
Once the distortions and displacements have been computed, texture synthesis occurs on the deformed patches with samples from the displaced texture swatches. At this point the graphcut algorithm [25] could cut a seam through the overlapping textured patches, but graphcut may not align all of the features in the synthesized textures.

We align these features with a deformation algorithm that resembles methods used in smoke animation [26]. First we blur the synthesized texture in the overlapping portions of both patches $P_1(x,y)$ and $P_2(x,y)$. For each pixel position $\mathbf{x} = (x,y)$ in the overlapping boundaries of the patches, we define a 2-D deformation vector $U(\mathbf{x})$, initialized to $(0,0)$. We then define an objective function

$$\varphi = k_1 \sum ||P_1(\mathbf{x}) - P_2(\mathbf{x} + U(\mathbf{x}))|| + k_2 \sum |\nabla \cdot U(\mathbf{x})| \tag{4.9}$$

to maximize the color match while minimizing the amount of deformation over the patch overlap area. We set $k_1 = 1, k_2 = 9$ and our RGB channels ranged from $0 \ldots 255$. Our feature mapping implementation computed $\partial\varphi/\partial U(\mathbf{x})$ and minimized φ using conjugate gradients. We found the deformation vector can be solved on a subset of the overlapping pixels and interpolated on the rest to accelerate convergence and further smooth the deformation, though at the risk of overlooking the matching of smaller features.

Once the deformation vectors have been constructed on the overlapping boundaries, they are blended into to the new patch's interior via Poisson image editing [27], and the graphcut algorithm is finally applied to find the optimal seam through the overlapping area. Our cost function for cutting a seam through overlapping areas is a weighted combination of pixel color and recovered surface normal, though color alone suffices in most cases. Our overlapping area is 16 pixels wide. Fig. 4.6 demonstrates the result.



(a) (b)

Figure 4.6: Brick texture without (a) and with (b) morphing.

4.3 Results

We demostrate several applications of our algorithm in photograph editing that would be difficult or at least cumbersome to achieve with current software.

Surface Replacement. These techniques were designed for the application of replacing the appearance of a photographed surface with that of a synthesized texture. This works best when the photographed surface is untextured and nearly Lambertian (e.g. skin, clothes, sculptures), illuminated by a single directional source. Errors in the recovered normal field can be rectified by additional user manipulation. Figs. 4.7, 4.10 and 4.11 give three examples of texturing different real world surfaces. Fig. 4.10 (f) pushed our method to its perceptual limits. The constant size and well-known shape of text characters, more so than the other textures, accentuate the inaccuracy of estimated normal field.

Detail Generation. Hand painting objects into a photo with plausible shading is not difficult, but painting detail into such artificial objects can be time consuming. In Fig. 4.8,

several vases are painted into a scene with a plausible approximation of shading, then different textures are applied to create intricate details that follow the surface implied by the painted shading.

Normal Transfer (Embossing). In Fig. 4.9, the recovered surface normals from one image is applied to another, yielding an embossed result. Poisson image editing is used to seamlessly merge transfered normal and brightness into the target image's normal and brightness respectively. Texture on the original surface in target image is extracted as texture swatch. If available texture is not large enough, a 2D texture synthesis may be applied to generate a larger one. A 256×256 pixel texture is enough for generating all results for this paper.

Lighting. Though the original photographed surface brightness can be used to shade the result, the synthesized texture with the recovered surface normals can be rendered under any desired lighting configuration. The synthesized texture can even be environment mapped to appear more naturally embedded in the scene as demonstrated in Fig. 4.8. An environment map can be synthesized from the background of the photograph through inpainting and extrapolation [28].

Performance. The synthesis speed highly depends on searching effort, patch size and whether displacement mapping and feature matching are enabled. When the texture has a large regular pattern like brick, searching a larger area is required to find a match. Some typical speed on a 1.2GHz Athlon is shown in Table 4.1.

Image	Displacement Mapping	Feature Matching	Time (Second)
Fig. 4.10(d)	Yes	No	55
(Fig. 4.10(d))	No	No	24
(Fig. 4.10(d))	Yes	Yes	140
Fig. 4.7	No	Yes	90
Fig. 4.11(c)	No	No	12
Fig. 4.8 (all vases)	No	Yes	63
Fig. 4.9(b)	No	No	18

Table 4.1: Features and running times of figures in this paper.

Figure 4.7: Brick texture follows the surface in the original photo (inset).

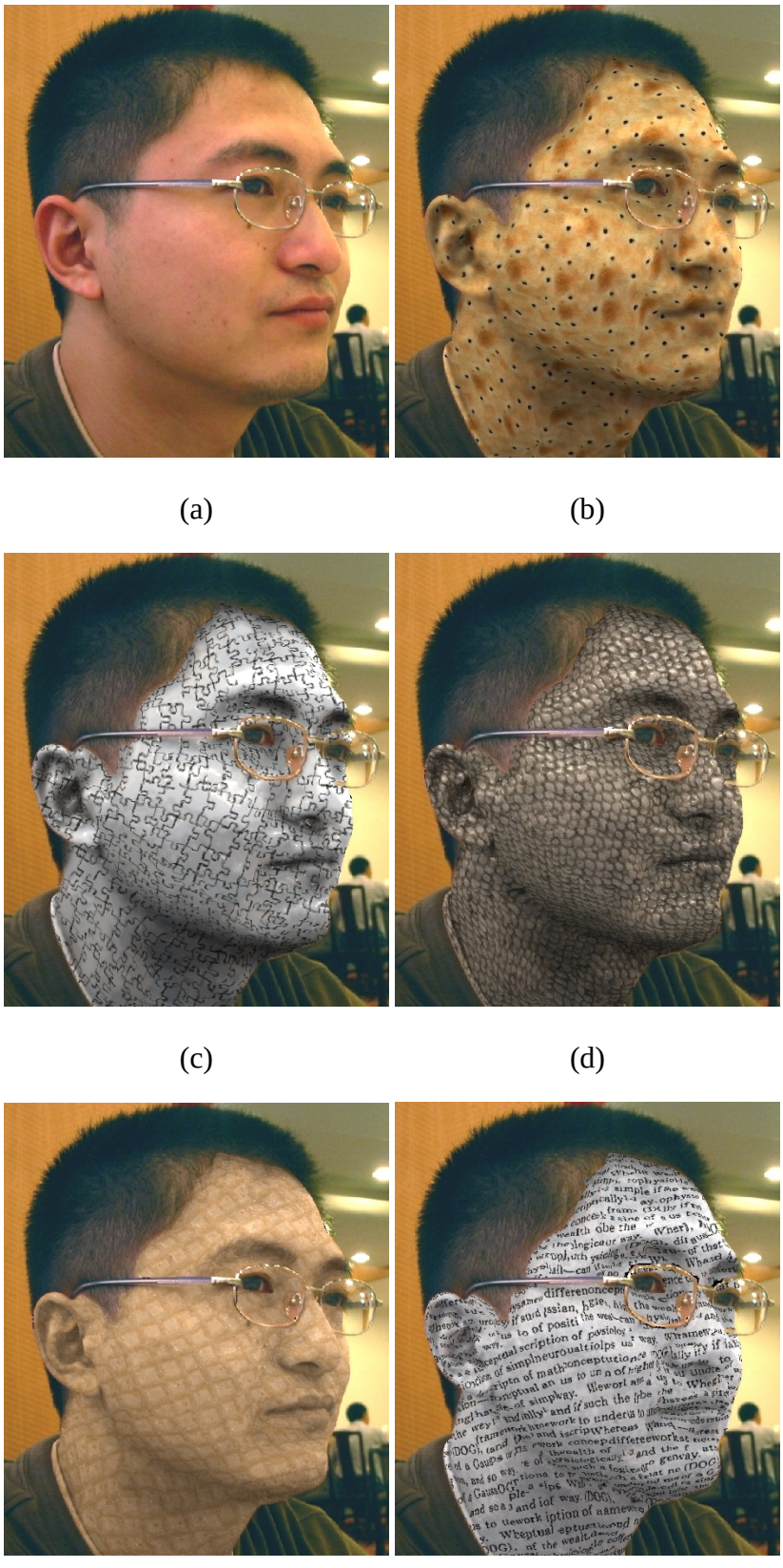
Figure 4.8: Texture synthesis (below) yields detail that follows the surface implied by the hand painted shading (above).

(a)

(b)

(c)

Figure 4.9: The normal field recovered from one image is transferred onto another.

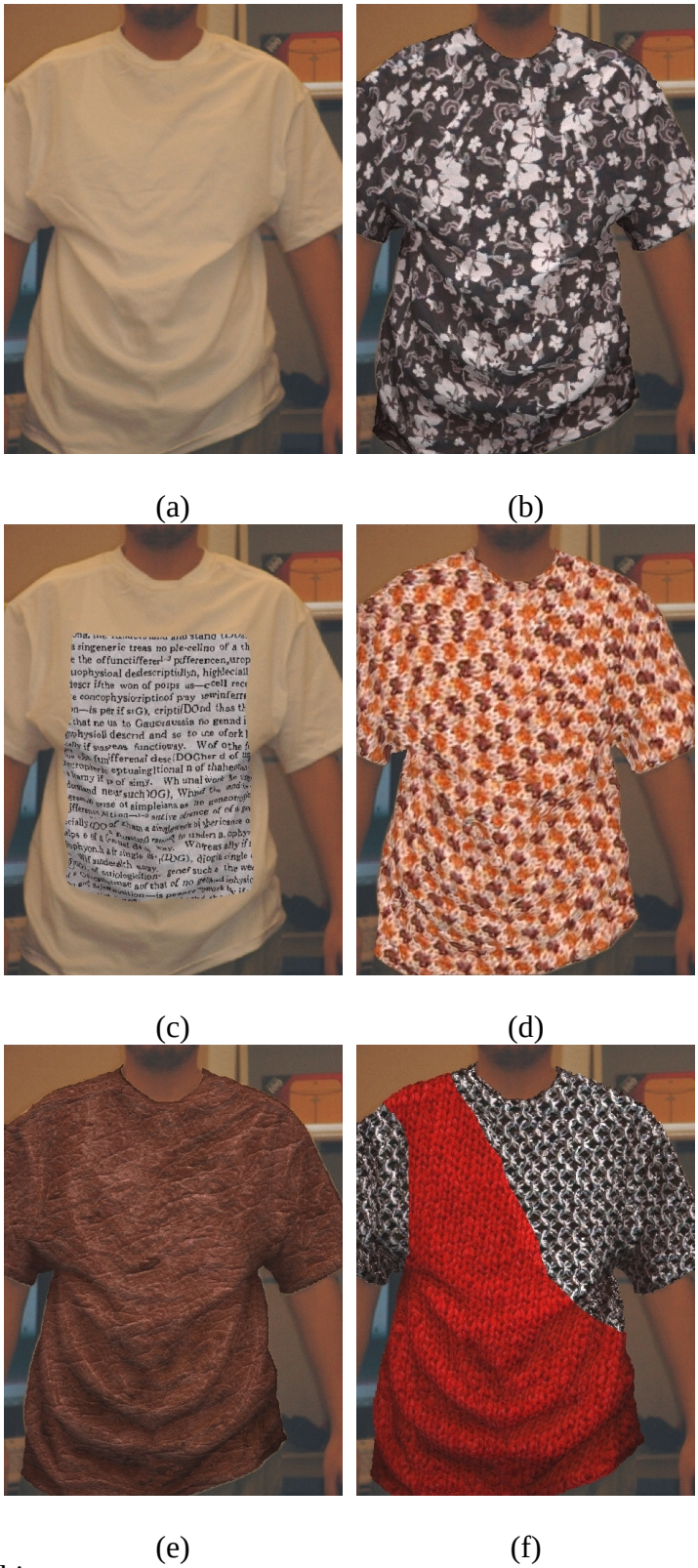


(a) (b)

(c) (d)

(e) (f)

Figure 4.10: Makeup.



(a) (b)

(c) (d)

(e) (f)

Figure 4.11: Fashion.

Chapter 5

RotoTexture Mapping

(a) (b) (c)

Figure 5.1: Fig. 1. TextureShop has already shown how to synthesize texture on a surface depicted in a photograph, such as replacing skin (a) with blue tile (b). RotoTexture adds the ability to synthesize time-coherent texture on a video sequence of a dynamic surface, such that the texture features in the first frame (b) correspond to those in a later frame (c). Rototexture also adds the ability to map an image onto the depiction of a surface in a photograph or video, demonstrated by the shirt's Da Vinci image whose deformation follows the wrinkles.

We propose a video editing system that allows a user to apply a time-coherent texture to a surface depicted in the raw video from a single uncalibrated camera, including the surface texture mapping of a texture image and the surface texture synthesis from a texture swatch. Our system avoids the construction of a 3-D shape model and instead uses the

recovered normal field to deform the texture so that it plausibly adheres to the undulations of the depicted surface. The texture mapping method uses the non-linear least-squares optimization of a spring model to control the behavior of the texture image as it is deformed to match the evolving normal field through the video. The texture synthesis method uses a coarse optical flow to advect clusters of pixels corresponding to patches of similarly oriented surface points. These clusters are organized into a minimum advection tree to account for the dynamic visibility of clusters. We take a rather crude approach to normal recovering and optical flow estimation, yet the results are robust and plausible for nearly diffuse surfaces such as faces and t-shirts.

5.1 Introduction

Disney's "Snow White," rotoscoping has allowed animators to capture the fluid motion of live-action video sequences, but with the novel appearance of a cartoon by overpainting the recorded motion with animated characters. Since then, a variety of motion capture tools have been developed that record the motion of an articulated figure (ranging from the poses of a body to the expressions of a face) so it can be reproduced with an altered appearance, as demonstrated in modern form by "The Polar Express."

One desirable way to alter appearance is to apply a new texture to a surface depicted in a video sequence, such as the example shown in Fig. 1. The ability to synthesize a texture or apply a texture image to a video sequence provides an alternative to the expensive, time consuming and uncomfortable special effects make-up that is common in science fiction and horror productions. Surface textures can also be applied to the video depiction of clothing, objects and buildings to customize their appearance without the expense of physically constructing the textured material or reshooting the scene. These methods are largely automated and rely on single-camera uncalibrated video, and so provide an attractive tool for personal digital content creation, such as the retexturing of faces to make home video

more interesting.

Methods exist that can texture a surface depicted in a single photograph [31; 32]. The TextureShop approach in particular avoided the need for a surface reconstruction and instead recovered a normal field that sufficed to deform a texture to make it appear to follow the undulations of the surface onto which it was superimposed. The task of texturing video challenges to this approach by requiring the texturing to be coherent over time, to prevent texture features from appearing and disappearing and to keep the texture perceptually fixed on the surface as it and the camera move.

Section 5.2 reviews existing methods that can extract the geometry and its motion from a video sequence that would allow its retexturing. These methods require calibration, multiple cameras and/or structured light though some, e.g. [33], can use the multiple views provided by an uncalibrated video sequence to reconstruct a static surface. Our goal is thus to texture a moving surface in an uncalibrated video sequence.

This paper describes a toolkit, called RotoTexture, consisting of two new methods for texturing a moving surface depicted by a raw video sequence. The first of these, RotoTexture Mapping, creates a temporally coherent mapping of a texture image onto the depiction of a moving surface, such that the texture image continuously deforms to follow the changing undulations of the surface. The second, RotoTexture Synthesis, maintains a temporally coherent collection of surface patches that allow TextureShop to texture the surface depicted by each frame such that the texture continuously follows the moving undulations of the surface.

RotoTexture Mapping, described in Sec. 5.3, improves TextureShop, which was limited to the construction of small cluster-based charts to support texture synthesis, by minimizing the energy of a rectilinear spring network to plausibly warp an entire texture image onto the surface depicted in a single frame. Frame-to-frame coherence is maintained by constraining nodes in the spring network to feature points in the video.

RotoTexture Synthesis, described in Sec. 6.1, supports the temporally coherent motion

of the pixel clusters used by TextureShop. Novel algorithms are developed to allow optical flow to advect these clusters while maintaining consistent texture coordinates within each cluster and between overlapping clusters. A new data structure called the Minimum Advection Tree determines how each cluster can be initiated at its most appropriate frame, and advected from there both forward and backward in time.

Results, presented in Sec. 6.2, are provided from a prototype implementation constructed using a simple optical flow interpolated from the sparse motion of tens of feature points. While these conditions lead to some texture swimming on static models, they demonstrate how well the method achieves the goal of texturing moving surfaces depicted in single camera raw video.

5.2 Previous Work

RotoTexture is an extension of Textureshop to video. Textureshop [31] describes a method of distorting a texture synthesis to follow the undulations of a surface depicted in a single uncalibrated photograph. The synthesized texture should appear as if it were applied to the surface and projected to the image, so the distortion is primarily the foreshortening of distance, and derived from a surface normal recovered via shape-from-shading. A retexturing method that recovers a surface model from a single photograph by analyzing its distortion of an existing regular texture is also available [32]. This analysis could also be extended to video, but would still require a pre-existing regular texture. RotoTexture Mapping and Synthesis both require the tracking of a small number of feature points which could be extracted from a regular texture, but need not be.

5.2.1 Shape From Shading

One could synthesize a coherent texture on the surface depicted in a video by reconstructing a 3-D meshed representation of the surface and performing texture synthesis on the mesh

[1; 2].

Shape from shading is a well studied area in computer vision that recovers a 3-D surface mesh from the sampling of an object's reflection recorded by image pixels [24; 34]. These methods have required at least one of multiple images, calibrated cameras and/or structured light for adequate reconstruction. One notable exception is Single-View Modeling [35], which can extract a free-form curved surface from a single photograph with the help of a sparse user-specified set of normals, silhouettes and creases. The multiple images drawn from a video from a single uncalibrated camera can be used to construct a decent 3-D model of a static object [33].

Dynamic objects, such as moving surfaces, pose a more challenging reconstruction problem. Zhang *et al.* [36] needed both multiple cameras and structured light to recover the shape and motion of a dynamic scene, but was effective enough to capture and reproduce the subtle geometry, appearance and motion of faces [37].

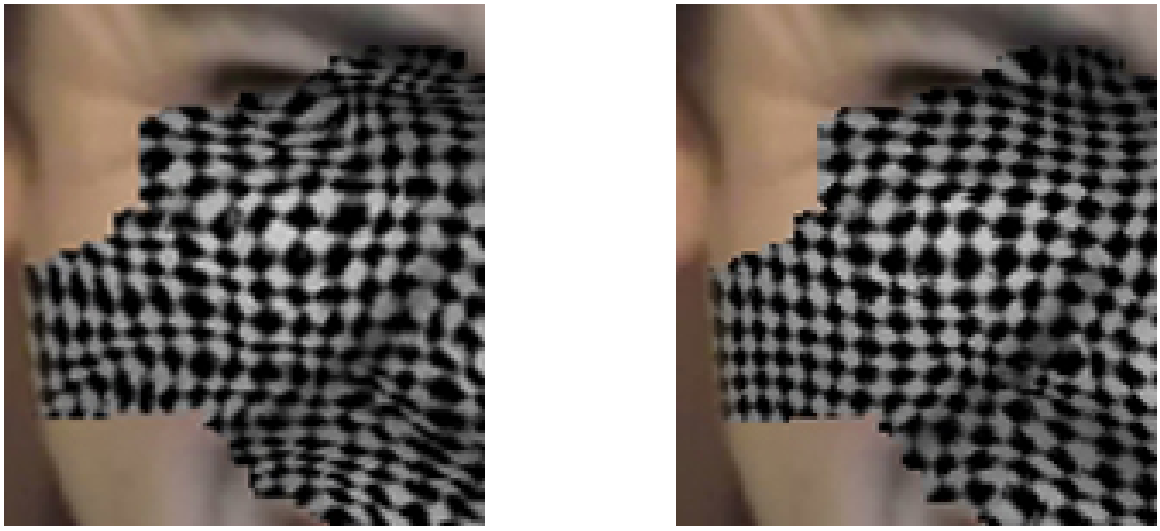
Some can recover 3D shape from a single camera given the assumption that the object is a combination of basis shapes [38] [39] [40]. Such approaches factor the tracking matrix to find both the motion and the deformation, but do not use the shading information for surface reconstruction and this low-frequency basis approach can overlook the high-frequencies of small surface details like the wrinkles important in recognizing facial expressions.

5.2.2 Optical Flow

Optical flow is a dense estimate of the relative motion between corresponding features and points of two images [41]. DeCarlo and Metaxes [42; 43] projected the optical flow onto the motion parameters of a dynamic face model to inhibit error and to detect and reproduce plausible expression. The spring model used for RotoTexture Mapping similarly restricts the behavior of the deformation of the texture image to the parameters of a flexible surface model. Our spring model is a non-linear least-squares fit, a common approach in vision for fitting geometry to image constraints, used here in a unique manner to match the

deformation of the image texture to the foreshortening predicted by the recovered normal field, and in a fashion that enables time-coherent animation of the texture.

Optical flow algorithms usually match sparse features between two video frames and interpolate this matching into a smooth dense vector field. The quality of optical flow depends on the distribution and accuracy of feature points. The criterion for a feature point can be relaxed until every pixel becomes a feature and the optical flow is a least-squares deformation from one image to the next. In any case, optical flow methods are not yet accurate enough to be able to deform the color signal produced by a texture synthesized or mapped in the first frame to frames in the remainder of a sequence, as demonstrated in Fig. 5.2.



(a) Optical Flow (b) RotoTexture Synthesis

Figure 5.2: (a) Optical flow can advect the image color signal from a texture on a surface, but suffers from numerical and resampling errors. (b) RotoTexture Synthesis advects image clusters corresponding to surface patches, and re-textures these clusters at each frame of the video.

For most surfaces, especially Lambertian ones, a change in surface shading implies a change in the surface orientation that can reveal further information on how the surface (and/or camera) moves. Our system combines both optical flow and the normal recovered by shape-from-shading in its estimation of surface motion.

5.3 RotoTexture Mapping

We want to deform a texture image over the image of a shaded surface so that the texture image appears to follow the undulation of the surface. We treat the texture image as an elastic membrane formed by a connected rectilinear network of springs. The surface normal recovered from the shaded image of the surface indicate how a texture image mapped onto it should be foreshortened. We set the desired length of these springs to a uniform fixed value, and

we initialize the length of these springs to form a rectilinear lattice over the original texture image, and solve for the deformation that minimizes the energy of this spring system.

Textureshop [31] defined a similar deformation by propagating inter-pixel distances to represent the distortion of foreshortening. Textureshop propagated these distances (and their orientations) across a small cluster of pixels with similar normals, but here we need to propagate these distances across an entire texture image. We use the spring network to restrict the behavior of this propagation, such that errors in the recovered normal and inconsistencies in the propagation are filtered out, yielding results that if not entirely accurate at least appear plausible for a flexible surface.

5.3.1 Surface Model

Let $U_i = (u_i, v_i)$ be one of a rectilinear 2-D grid of nodes evenly spaced across the texture image T, and let $X_i = (x_i, y_i)$ indicate its rendered destination on the screen. Our goal is to find the screen positions X_i of the rectilinear grid nodes that cause them to appear to be uniformly spaced across the underlying surface displayed on the screen, as illustrated in Fig. 5.3.

Let $X_{ij} = X_j - X_i$ be a vector from the screen position X_i of node i to the screen position X_j of a neighboring node. Recall that TextureShop derived an operator, P, that used the recovered normal to project a screen vector, e.g. X_{ij} onto the surface [31]. Let N_i and N_j

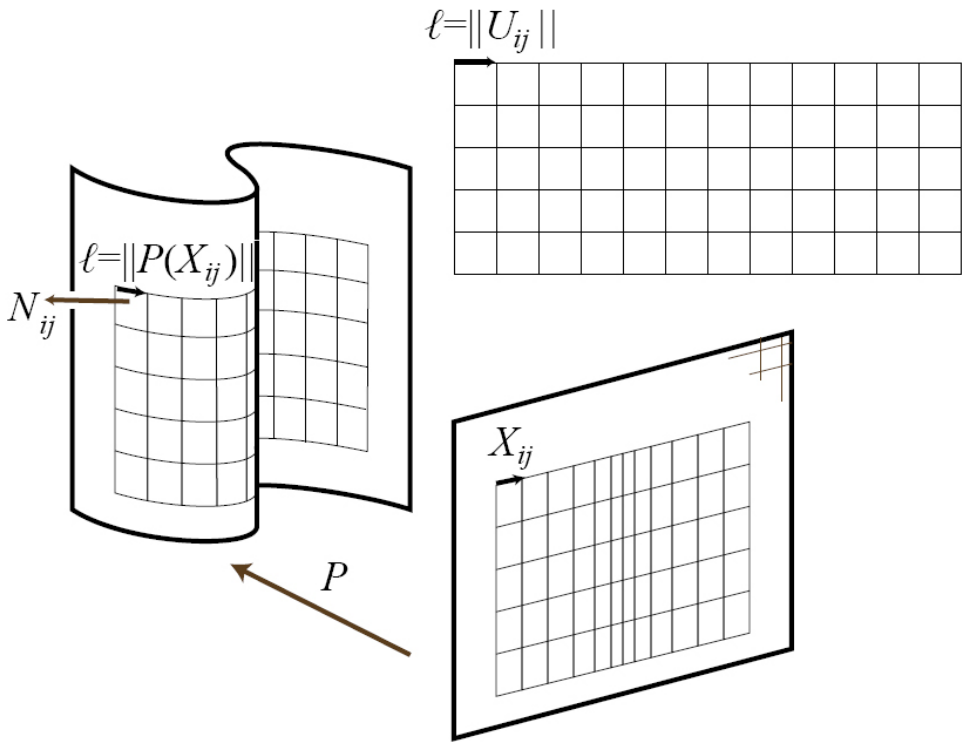


Figure 5.3: RotoTexture Mapping coordinates. Our goal is to create the appearance (lower right) that we have mapped the texture grid (upper right) isometrically onto the surface (left). This mapping is defined by solving for the vectors X_{ij} between neighboring vertices of the screen projection of the texture grid that project back to uniform length vectors $P(X_{ij})$ on the depicted surface.

be the recovered normals nearest to screen positions X_i and X_j, respectively, and let $N_{ij} = (N_i + N_j)/||N_i + N_j||$ be their average. This average normal N_{ij} allows us to define $P(X_{ij})$ as the surface projection of the screen vector X_{ij}, and the ratio of the lengths $||X_{ij}|| : ||P(X_{ij})||$ indicates the texture distortion due to foreshortening.

For simplicity, we will assume an isometric surface texture mapping, such that the length $\ell = ||U_{ij}|| = ||P(X_{ij})||$ for all nodes i and their neighbors j. Our goal is thus, given the recovered normals N_i, to find X_i such that $||P(X_{ij})|| = \ell$. To this end we seek to minimize the total energy $\sum E_{ij}$ of the spring system

$$E_{ij} = E_{ji} = (P(||X_i - X_j||) - \ell)^2. \tag{5.1}$$

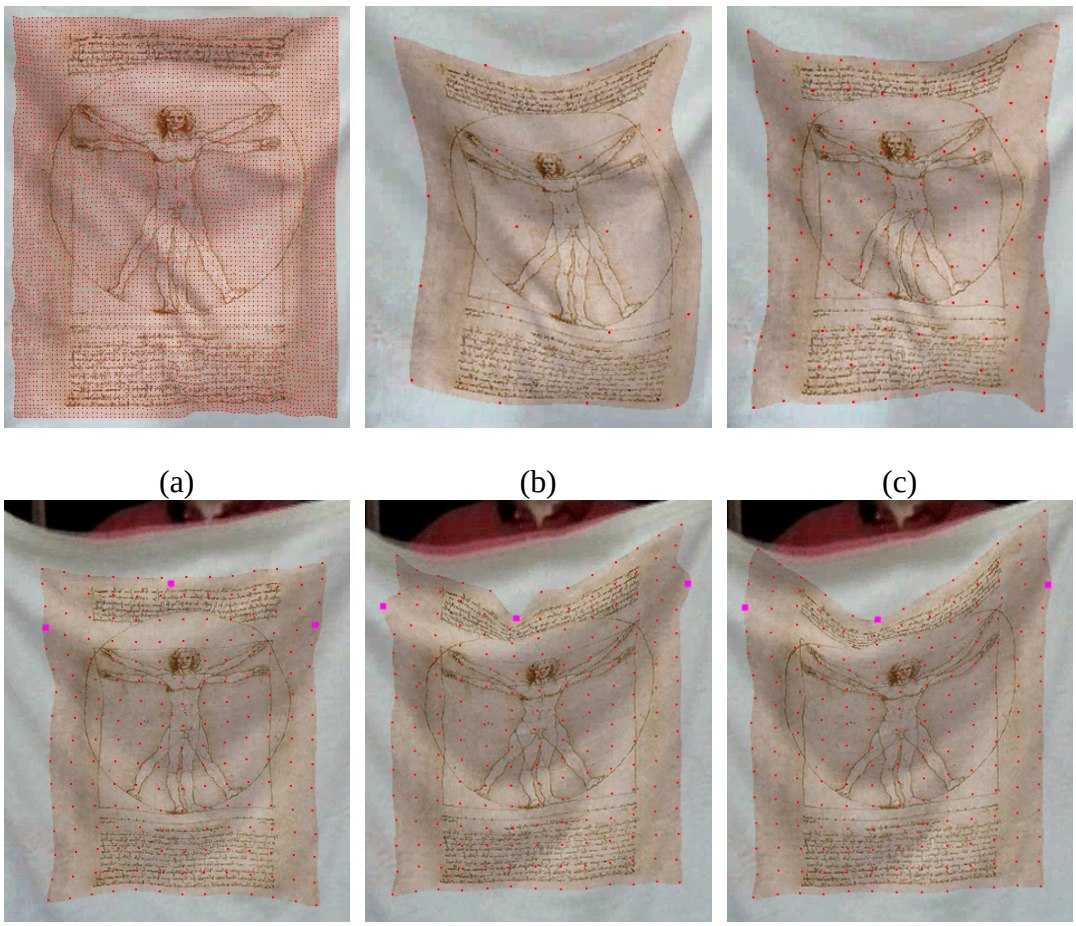
Since the solution positions $\{X_i\}$ affect the measurement of normals $\{N_i\}$, this system is a non-linear least-squares problem, which we solve by gradient descent.

5.3.2 Coarse Grid Solution

At finer resolutions the total energy landscape $E[\{X_i\}] = \sum E_{ij}$ has many local minima that hinder global minimization, as shown in Fig. 5.4(a-c). A multiresolution approach avoids these local minima pitfalls by reducing the number of parameters over which to minimize the energy system. We reformulate the texture mapping as a piecewise affine warp controlled by a coarser grid of solution points $\{\hat{X}_i\} \subset \{X_i\}$, but still measure the total energy at the finest resolution $\{X_i\}$. This leads to a multiresolution relaxation where a coarse grid solution initializes a fine grid solution. We found a two-stage relaxation sufficed, consisting of a coarse grid of 32×32-pixel cells and a fine grid of 6×6-pixel cells.

5.3.3 Feature Points

For a static image, the energy minimization produces a convincing distortion of an image texture so it appears to adhere to the underlying surface. For a coherent sequence of images,



(a) (b) (c)

(d) (e) (f)

Figure 5.4: (a) The optimizer gets stuck in local minimum when control point spacing is 4 pixels. (b) A control point spacing of 90 pixels oversmooths details of the normal field. (c) We obtained the best result for a control point spacing of 32 pixels. (d) An image pasted onto a surface with three feature points. (e) Red dots shows control points every 24 pixels, exhibiting distortion. (f) Additional weighting during optimization eliminates this distortion.

errors in temporal and spatial sampling, normal estimation and warp reconstruction accumulate unwanted translation, rotation and other effects in the warp that cause the image to appear to "swim" on the underlying surface. It is therefore necessary to fix the position and orientation of the image on the surface through the identification and tracking of a minimal collection of surface feature points.

Feature points are integrated into our model by identifying a control node in our mesh with each feature point as shown in Fig. 5.4(d). Let F_k be a feature point and let X_k be its corresponding control point. Then the added energy penalty incurred by X_k when it strays away from F_k is proportional to the distance

$$E_k = \alpha ||X_k - F_k||, \tag{5.2}$$

where the penalty strength $\alpha = 50$ in our implementation.

This simple penalty constraint can cause unnatural distortion artifacts as shown in Fig. 5.4(e). We can reduce these artifacts by smoothly extending the penalty constraint to a neighborhood $\{X_j\}$ of nodes near X_k. We find the desired positions for the $\{X_j\}$ given that X_k should be at F_k with a separate optimization that assumes there is only one feature point F_k and records the resulting positions of the neighborhood nodes $\{X_j\}$ as $\{F_j\}$. We then penalize the positions of the $\{X_j\}$ toward these $\{F_j\}$. in the original optimization that includes the original feature points. The weights of these penalties should taper off gradually with distance from the original feature point F_k as

$$E_j = \alpha \exp\left(\frac{-||F_k - F_j||}{\sigma^2}\right) ||X_j - F_j||, \tag{5.3}$$

where σ is 25% of the distance between feature points. The result yields the plausible solution shown in Fig. 5.4(f).

5.3.4 Temporal Smoothing

Since each frame is computed independently, except for the coherence of the feature points constraints, rapid changes in the recovered normal can lead to inconsistencies and visual

noise that can be reduced by a temporal smoothing of the texture mapping. We smooth the mapping $X(t) = \{X_i(t)\}$ at frame t with a partial Laplacian filter

$$X(t) \mathrel{+}= \frac{1}{2} w \left(X(t - \Delta t) - 2X(t) + X(t + \Delta t) \right) \tag{5.4}$$

using a filter weight $w = 0.1$.

5.4 Results

The motion of a cloth is captured with a video camera in Fig. 5.5. An image is pasted onto it with the technique described in Sec. 5.3. Three feature points are tracked on the surface and are used as constraints for the optimization to prevent the texture image from swimming on the surface. During optimization, five iterations are performed at 32 pixel control points spacing, then at 6 pixel spacing. The running time averages 110 seconds per frame.

(a) (b)

(c) (d)

(e) (f)

Figure 5.5: Pasting an image on a surface.

Chapter 6

RotoTexture Synthesis

6.1 RotoTexture Synthesis

TextureShop clustered pixels of similar recovered normal to reduce variation within each cluster and thereby reduce error when propagating texture coordinates from the cluster center to its boundary. But a dynamic surface will yield different recovered normal fields leading to a different arrangement of clusters from frame to frame. We assume the depicted surface, while dynamic, undergoes a motion that is mostly rigid-body and otherwise deforms in a subtle and localized manner. For example, the motion of a face follows the orientation of the head but also contains expression. The clusters are intended to correspond to patches on the surface, and though their image may move and change size, the relative shape and organization of clusters should remain consistent during surface motion.

TextureShop clustered pixels in a still image by like normal. Let C_{ij} denote the pixels $0 \leq j < |C_{ij}|$ in cluster i. For each cluster i, let $U_i : (x,y) \mapsto (u,v)$ describe the parameterization generated by TextureShop for that cluster that distorts the synthesized texture according to the foreshortening derived from the recovered surface normals. When applied to a sequence of video frames, the recovered normals of a dynamic surface change, and the clusters they yield may not correlate with clusters from neighboring frames.

The application of TextureShop's clustered texture synthesis to video requires the con-

struction of a time-coherent clustering. RotoTexture Synthesis uses an optical flow to advect clusters, which allows the clusters to evolve as the surface and view evolve while retaining their grouping of like normals in a temporally coherent manner.

6.1.1 Cluster Repositioning

An optical flow $O_{t_0 \to t_1} : (x,y) \mapsto (\Delta x, \Delta y)$ is a two dimensional velocity field of two-vectors that describes for each pixel $(x,y) \in I(t_0)$ its location $(x+\Delta x, y+\Delta y)$ in a new frame $I(t_1)$.

A number of techniques exist for recovering an optical flow from a video sequence. Since we have already organized the image into clusters corresponding to space-coherent surface patches, a coarse approximation of the optical flow generated from a relatively small number of feature points sufficed. Let $F_j(t)$ indicate the position $(x,y) \in I(t)$ in the frame at time t of feature point j. The motion of these feature points $\Delta F_k(t) = F_k(t+\Delta t) - F_k(t)$ yields a sparse 2-D vector field that when interpolated, e.g. using multilevel free form deformation [44], generates a coarse but adequate approximation of the optical flow.

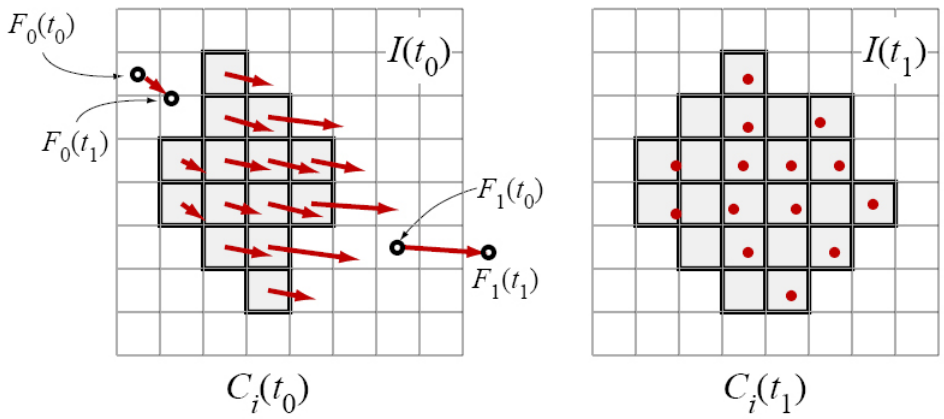


Figure 6.1: Optical flow (red arrows left) interpolated from feature points F_0 and F_1 (circles left) used to advect cluster pixels $C_i(t_0)$ into positions (red dots, right) interpolated into a new cluster $C_i(t_1)$.

As shown in Fig. 6.1, we move the pixels in clusters $C_{ij}(t)$ through Lagrangian advection under the optical flow $O_{t_0 \to t_1}$ into the image $I(t_1)$. The new cluster pixel positions

$O_{t_0 \to t_1}(C_{ij}(t_0))$ in general do not fall on pixel centers, so pixels in $I(t_1)$ are classified into the cluster $C_{ij'}(t_1)$ by their nearest neighbor $O_{t_0 \to t_1}(C_{ij}(t_0))$.

6.1.2 Cluster Reparameterization

We use the optical flow advection to propagate the pixel clusters from one frame to another. The new frame then contains clusters that need to be reparameterized to reflect the foreshortening distortions of its new field of recovered surface normals. The TextureShop method propagates a parameterization from a cluster center to its boundary, so the texture coordinates generated on the boundary of a cluster in the new frame with its new normals can differ significantly from the coordinates generated on the cluster's boundary in the previous frame. Since the cluster boundary needs to blend nicely with the neighboring cluster, changes in texture at the boundary are particularly noticeable.

We run TextureShop on clusters in the starting frame and find the seams of minimum color difference in the overlapping region between neighboring clusters via Graphcut. Our goal is to reparameterize a cluster in a subsequent frame while retaining its original texture coordinates along this seam. This maintains the color match between overlapping clusters during advection.

Let $B_{ij}(t) \subset C_i(t)$ be the pixels, indexed by $0 \leq j < |B_{ij}(t)|$, on the seam of cluster i at time t. We use the optical flow to advect cluster $C_i(t_0)$ to $C_i(t_1)$ and this advection takes each boundary pixel $B_{ij} \in C_i(t_0)$ to the position $O_{t_0 \to t_1}(B_{ij})$ in the frame at t_1. We then define a parameterization correction vector for each of these points j in each cluster i as

$$\Delta UB_{ij} = U(B_{ij}) - U(O_{t_0 \to t_1}(B_{ij})), \tag{6.1}$$

the difference in the desired texture coordinate of the original cluster boundary pixel $U \circ B_{ij}$ and the texture coordinate generated by TextureShop using the new normal field $U \circ O_{t_0 \to t_1}(B_{ij})$. Since $O_{t_0 \to t_1}(B_{ij})$ may not correspond to a pixel center in $I(t_1)$, its texture coordinates $U \circ O_{t_0 \to t_1}(B_{ij})$ may need to be interpolated from the texture coordinates of its

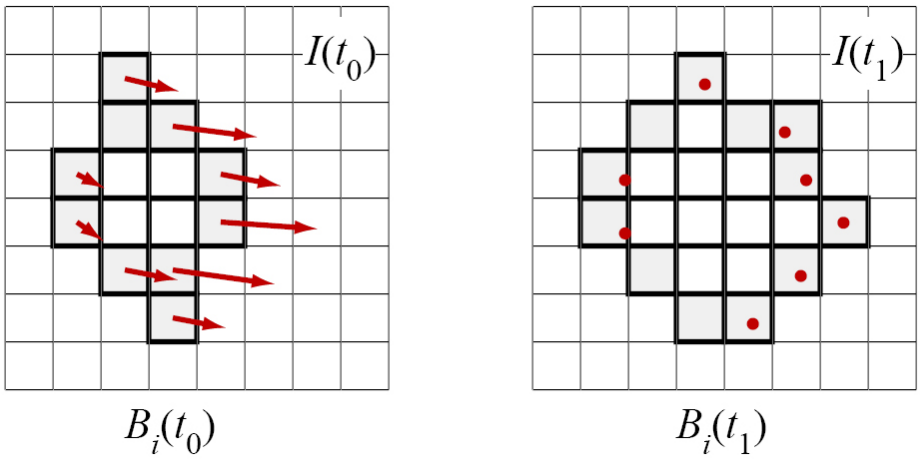


Figure 6.2: Boundary pixels at time t_0 (shaded left) advect into positions at time t_1 (red dots right) that preserve their texture coordinates, which are then resampled to correct the texture coordinates of boundary pixels $B_i(t_1)$ and eventually the entire cluster $C_i(t_1)$.

nearest four pixels in $C_i(t_1)$. We used nearest neighbor interpolation.

Likewise, the feature points $F_k(t)$ that generate the optical flow were chosen because they are easy to identify visually in each frame. While we want to prevent the appearance of texture swimming at any point on the displayed surface, we are especially sensitive to deviations in the texture at these feature points. We similarly define a parameterization correction vector for these feature points as

$$\Delta UF_k = U(F_k(t_0)) - U(F_k(t_1)), \tag{6.2}$$

the difference between the original desired texture coordinates of a feature point from frame t_0 and the texture coordinates generated by the new normal field at frame t_1.

We correct the parameterization U_{t_1} generated by the surface normals N_{t_1} recovered from $I(t_1)$ using a correction field constructed by interpolating the boundary and feature parameterization correction vectors. Let $\Delta U_{t_1} : (x,y) \mapsto (\Delta u, \Delta v)$ be the parameterization correction field constructed by interpolating the sparse correction vectors ΔUB_{ij} and ΔUF_k. This field corrects the parameterization at frame t_1 as

$$U_{t_1} \mathrel{+}= \Delta U_{t_1}. \tag{6.3}$$

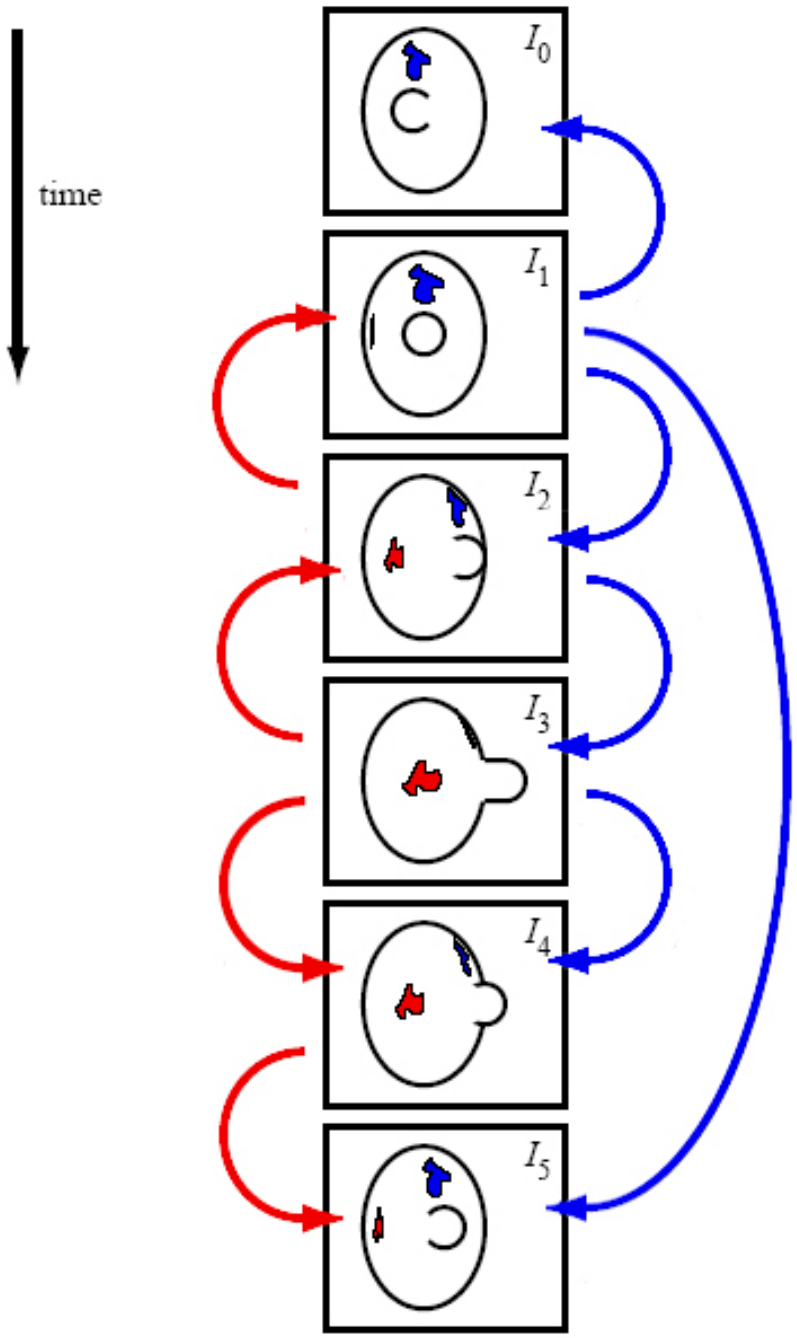
The parameterization correction terms are applied at the expense of the magnitude of the effect of foreshortened texture distortion. While the human perceptual system uses texture in part to resolve perspective, small errors on a non-simple surface can be perceptually insignificant, and in any case are a rather small price to pay for the more critical effect of temporal coherence of texture features.

6.1.3 Temporal Smoothing

Clustered texture synthesis, even when corrected by locking the texture coordinates at boundary pixels and feature points can still appear noisy because the normal field upon which they are built is not temporally smooth. We further stabilize the synthesized texture on the perceived surface by restricting the texture reparameterization and correction process to "key" frames (sampled typically every five frames) and interpolating the texture coordinates for the intermediate frames. This reduces oscillations and they more subtly blend into the actual motion of the surface. Since the texture clusters are advected every frame from an optical flow constructed from feature points and the per-cluster texture parameterization is interpolated between key frames, the reconstructed normal field directly influences the texturing of the key frames, but does not directly influence the clustering and parameterization of the intermediate frames.

6.1.4 The Minimum Advection Tree

Due to occlusion, parts of the surface may disappear and reappear when the video contains motions as simple as rotation. In such cases the optical flow advection alone cannot manage the disappearance and reappearance of a cluster corresponding to a given portion of the surface. In these cases, it is better to perform non-linear optical flow and cluster advection. Each cluster is constructed and parameterized in the frame where it most squarely faces the camera. The cluster can then advect and propagate its parameterization to the rest of frames.



(a)

Figure 6.3: A minimum advection tree for two clusters in a six frame video. The blue cluster in frames I_0 and I_5 is advected from the blue cluster root frame I_1. The red cluster does not even appear in the first frame of the video. The red clusters are advected from the root frame I_3.

The *Minimum Advection Tree* (MAT) is a directed graph that indicates for each frame the frames other than itself and its parent that are more similar to it than any other. We then compute optical flow, cluster advection and reparameterization from the root of this tree to its leaves, in an order that prioritized spatial instead of temporal coherence (e.g. frames at two different times may be very similar).

Ideally, a separate minimum advection tree is constructed for each cluster, and each cluster is advected independently, but this individual processing of clusters is expensive and memory incoherent. In practice it was more efficient to group clusters facing similar direction and process these "superclusters" together.

Furthermore, a "collision" in cluster shape can occurs when two different cluster advection paths lead to frames neighboring in time, and the accumulated error due to the different optical flows of the two paths causes a cluster to advect into different shapes. We were able to smooth this collision by advecting the cluster from one path backwards through the history of the other path and averaging the shapes.

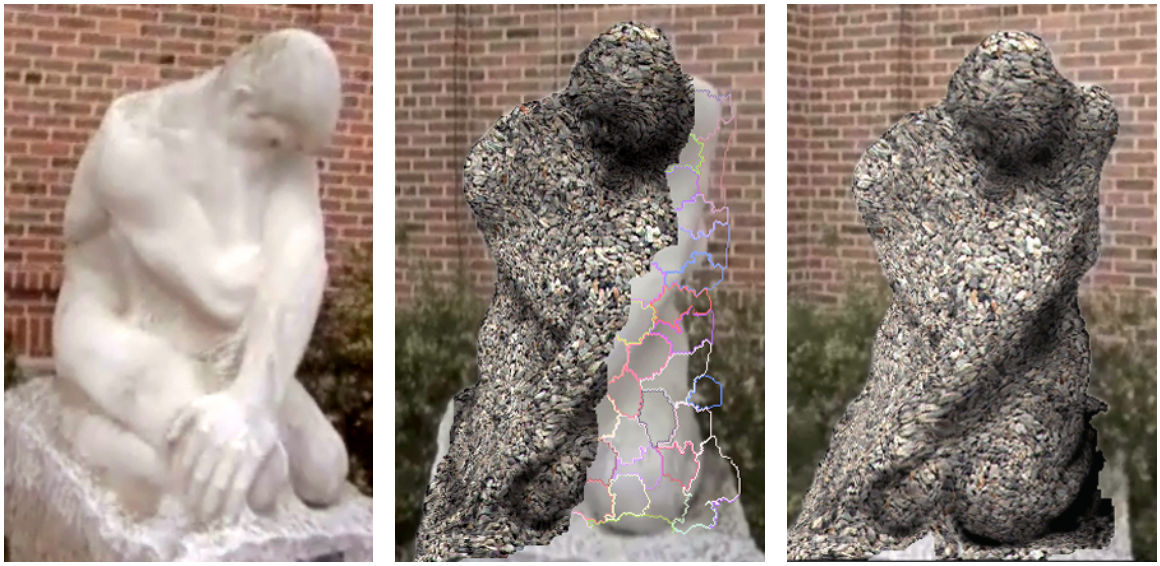
Costs are assigned to all advections. In our experiments, we assign the cost of the jump advection to non-neighboring frames four times as high as advection between neighbouring frames to reduces "collisions." Thus advection to a non-neighboring frame only made sense for distances larger than four frames in the past or future. Under this constraint, most video yields a MAT structure consisting of of a few long time-linear sequences.

To build a MAT rooted at a certain frame, any other frame is linked to that frame through a series of advections with lowest cost.

6.1.5 Rendering

Image brightness is used to modulate the diffuse reflection of the synthesized texture. The synthesized texture is rendered with a specular reflection based on the synthesized texture's normal oriented relative to the recovered normal field.

Graphcut [25] is used to find the optimal seam between clusters in an initial frame.



(a) (b) (c)

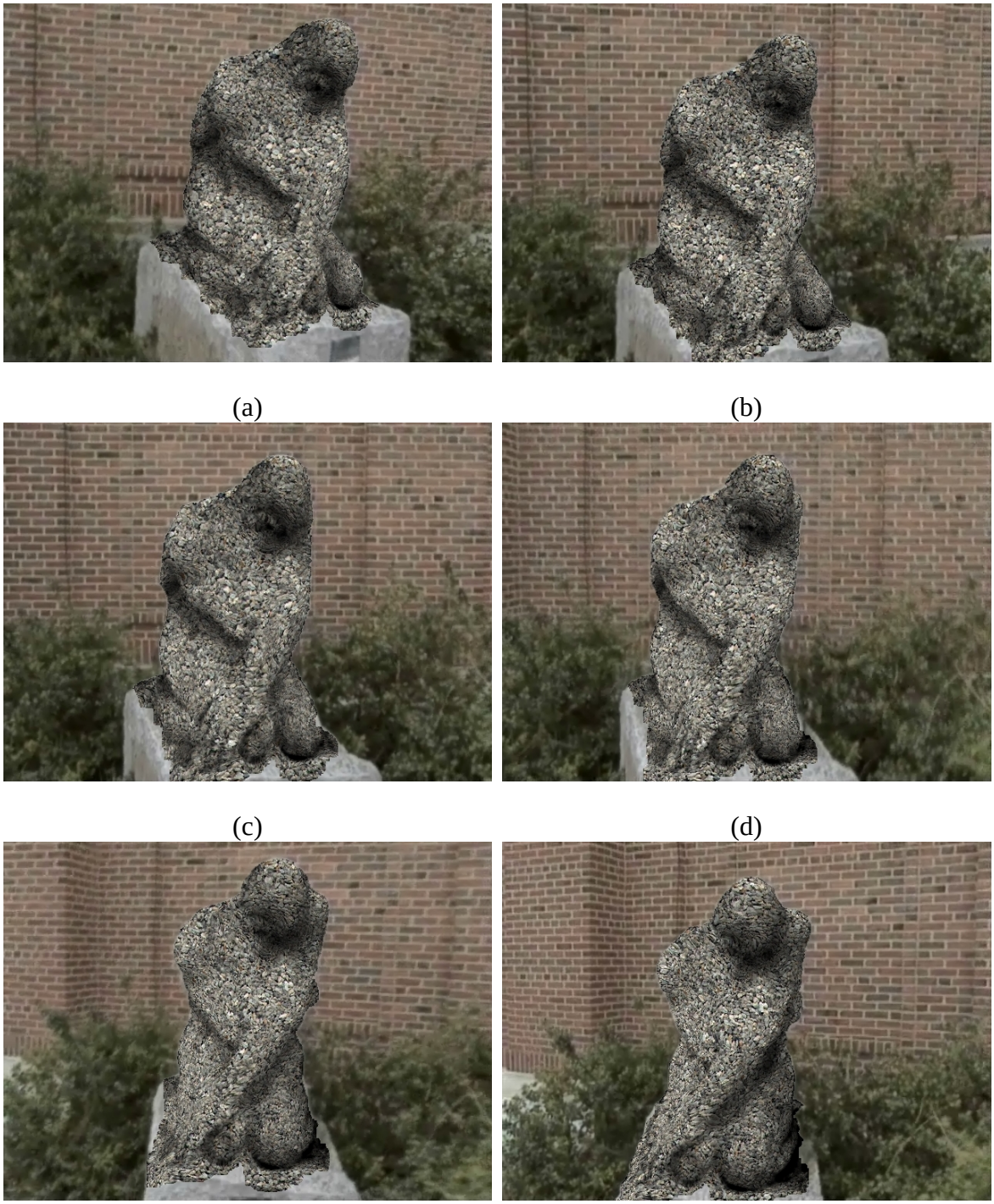
Figure 6.4: Parts of the statue is not visible from its initial pose (a). New clusters are generated at a later moment (b) and advected with MAT to cover the whole surface(c).

Subsequent frames retain this seam because the texture coordinates of cluster boundaries are retained during advection. However, we execute a 3-D extension of Graphcut over the time-space volume of clusters to improve this boundary (similar to [45]), using a roughly six-pixel-wide region surrounding the original advected seam.

6.2 Results

In Fig. 6.5, a statue is scanned with a handheld video camera. Fig. 6.4 shows frames from an arc of video frames about that statue. Most of the clusters are visible in the first frame, where they are defined and parameterized, and advected in forward time order as a single supercluster. The clusters not visible in the first frame are defined and parameterized in the final frame, and advected again as a single supercluster, in reverse time order. The synthesis is run twice for two those superclusters, and the overall synthesis time is 59.5 seconds per frame.

In Fig. 6.6, a total of 27 feature points are located and tracked on the face, some are automatically placed and tracked at easily detectable features while others are manually



(a) (b)

(c) (d)

(e) (f)

Figure 6.5: Texture on a statue.

placed and tracked on smooth but important locations, such as cheeks and nose tip. The optical flow is generated from these feature point correspondences by a free form deformation field. The absolute accuracy in the locations of the feature points is not essential, but jumps in their locations can generate high frequency oscillation in the resulting texture. We smoothed the location of the feature points using the partial Laplacian filter described in Section 5.3.

Fig. 6.8 and Fig. 6.7 show that our algorithm handles large face deformation and rotation robustly. The center sequence “blue” rotating face, the clusters are grouped into three superclusters at different stages of the rotation. Each supercluster is synthesized independently and their results are merged. The averaging synthesis time for these three sequences was 32.3 seconds per frame for the left “talking” sequence, 71 seconds per frame for the center “looking” sequence and 31.8 seconds per frame for the right “expression” sequence.

Each of these examples relied on a similar level of user interaction. The portion of the frame to be retextured was selected manually using Lazy Snapping[46], though could be isolated automatically with existing video matting techniques [47; 48]. The sparse sets of feature points in the initial frame of each sequence were picked by a combination of corner detection and manual selection, and tracked by simple block matching which was corrected manually when it failed. The remaining tasks executed automatically.

6.3 Conclusion

The limited application of texturing an animated surface allows us to avoid the need for accurate optical flow and full shape from shading that has otherwise occupied the attention of much work in the vision and graphics communities. RotoTexture generated the results by tracking about 30 feature points, and with inaccurate, locally recovered normals assuming simple Lambertian reflection. The added robustness of optical flow advection of space-coherent clusters coupled with dependence only on the normal field instead of a full 3-D

(a) (b)

(c) (d)

(e) (f)

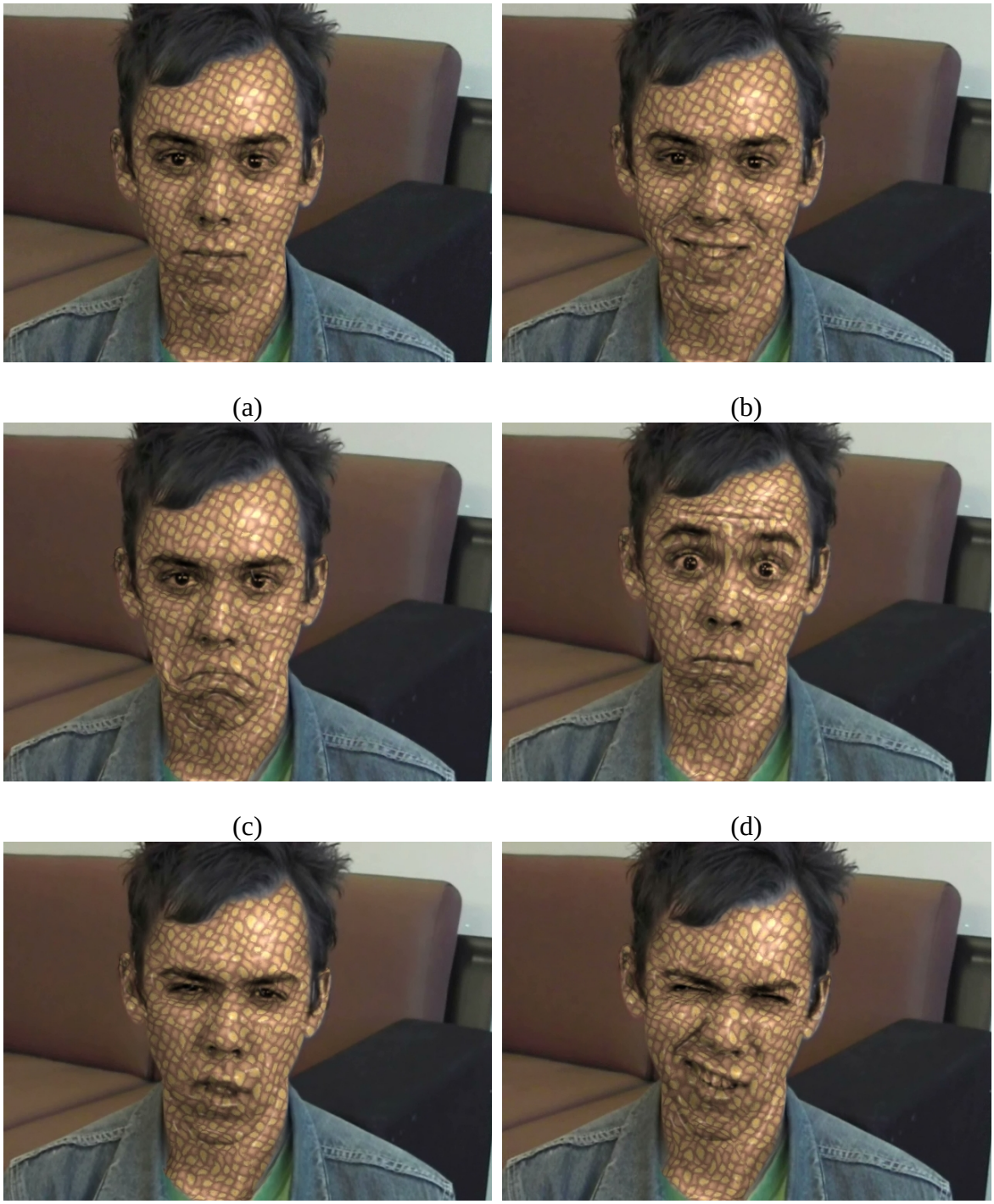
Figure 6.6: RotoTexture Synthesis on a talking face.

(a) (b)

(c) (d)

(e) (f)

Figure 6.7: RotoTexture Synthesis on a rotating face.



(a) (b)

(c) (d)

(e) (f)

Figure 6.8: RotoTexture Synthesis on a deforming face.

shape representation yields a tool that works sufficiently (and surprisingly) well given the raw video from a single uncalibrated camera.

Maintaining temporal coherence in the synthesized videos posed a significant challenge, and despite our best efforts at tracking features and smoothing the results, the texture can still jiggle and swim slightly across the surface. The primary source of this swimming is the inaccuracy of feature points tracking. The simple block-matching method we used fails, especially on smooth surfaces like the white sheet in Fig. 5.5 and the statue in Fig. 6.5.

A secondary source of swimming artifacts resulted from discrete sampling. We used the nearest neighbor to interpolate advected texture coordinates to parameterize clusters, and this choice resulted in small sub-pixel errors that accumulated to exhibit swimming artifacts. Supersampling both the image and the texture would likely reduce such artifacts.

Our target application of texturing moving surfaces, such as that of a talking face in Fig. 6.6, 6.7 and 6.8 fortunately obscures these swimming artifacts. The artifacts are much more obvious on a stationary surface, such as the statue in Fig. 6.5, where other techniques can reconstruct (and hence texture) a complete 3-D model [33].

Chapter 7

Detail Preserving Image Morphing

Morphing is a common practice in digital photograph editing, but morphing does not replace the detail lost where the image is enlarged. We propose an image editing system that decouples feature position from pixel color generation to achieve a morph that preserves texture detail and orientation near the dragged silhouette, synthesized using the original image as an anisotropic texture. We introduce a new distortion to patch-based texture synthesis that aligns texture features with image features. A dense correspondence field between source and target images generated by the control curves can then guide texture synthesis.

7.1 Introduction

We propose a novel image editing system that allows a user to select several control curves based on an image's features and moves them to generate a new image whose details are resynthesized to accommodate the new feature locations. Traditional image morphing interpolates the image color signal, which blurs details if the morphed image region dilates.

We decouple the deformation of images features from the generation of the new color signal, and inserts an anisotropic patch-based texture synthesis step between to preserve detail in the interpolation over missing pixels in the target image. Our patch-based texture

synthesis approach resembles that of Graphcut Textures [25] with an additional distortion to the texture coordinates for each patch to align the target image features with the user specified morphing effect.

Our approach assumes that the texture orientation of the image is relevant to the curvature of the nearby feature lines, such as the hair on a bunny's ear in Fig. 7.1. When texture orientation is not related to feature lines, such as the grassy isotropic background texture, we separate anisotropic and isotropic texture synthesis, and combine the results using the feature curve as a matte.

The key contribution of this new detail preserving morphing is the novel ability to deform the feature lines of patch-based synthesis to fit a global feature line specification. Local synthesis methods like Image Analogies [49] can synthesize a texture to adhere to a given feature line, but its per-pixel synthesis yields more high-frequency noise than more preferable patch-based synthesis approaches. Image Quilting [45] included examples where the silhouette of the image of an orange was filled with different textures, by iterating on progressively smaller patches. Lacking the ability to create a new feature slope from source feature lines, it generated small repetitive patterns near the boundary, because all of the patches with a certain slope came from the same location in the source image. A previous feature matching approach distorted each patch to connect its internal feature lines with those of neighboring patches [50], but these patch-internal feature lines were not fit to a global feature line to an arbitrary shape.

7.2 Previous Work

Texture synthesis generates a new texture from a sample texture swatch. Local approaches generate each pixel independently from others, which suffers from high frequency noise and loss of structure [51; 52]. Patch-based texture synthesis better reproduces texture appearance by seamlessly pasting together patches of the original texture [25; 45], and our

contribution is built on this technology.

Our application follows in the tradition of other novel image editing metaphors enabled by texture synthesis, including Image Analogies [49] and Texture by Numbers [53].

Others have also incorporated user control in texture synthesis, such as the specification of a feature map [50], or a guidance vector field [54]. The motion of our before-after control curves establishes such a guidance vector field, but we pay additional attention to the behavior of the texture synthesized around the control curve.

Textureshop retextured images using recovered normals to foreshorten the synthesized texture [31], including embossing examples whose texture source was the image itself distorted by a new normal field. Our contribution likewise uses the image as the texture source, but instead distorts the texture synthesis by the motion of the control curves.

Image Completion also propagated an image's textures [55], but was limited to the linear motion of isotropic textures, which is extended by the arbitrary shaped motions and patch-based synthesis of our contribution. Similarly, Object-Based Image Editing manipulates objects in a photo as a collection of small regions [56], but does not deform texture at pixel-level detail.

7.3 Feature Aligned Cluster Parameterization

Deformation Field. We seek to morph a source image $I(x,y)$ to generate a target image $I'(x,y)$. The user selects several control curves $F_i \subset I$, which can be conveniently but not necessarily chosen from feature lines in I, and manually deforms them into $F_i' \subset I'$ to indicate the desired morphing effect. For each pixel of the target control curve $p_f' \in F_i'$, let $p_f \in F_i$ denote its preimage under the induced deformation. We construct a smooth deformation field $D : I \to I'$ by solving Laplace's equation

$$\nabla^2 D(x,y) = 0 \tag{7.1}$$

with the boundary conditions $D(p'_f) = p_f - p'_f$ and $D = 0$ on the image border, though to hasten the solver, we can set $D = 0$ beyond a given radius about F'_i.

Tangent Fields. We similarly sample and interpolate an initial tangent vector field $T : I \rightarrow S^1$ initialized with the tangents of the control curves F_i, and a target tangent vector field $T' : I' \rightarrow S^1$ from F'_i. These tangent fields will later orient the synthesized texture with respect to the feature curves.

Feature Magnitude Fields. During synthesis, we prefer to start at places with the strongest features, then fill the gaps between features whose texture is relatively isotropic. To facilitate such an approach, we define an initial feature magnitude field $M : I \rightarrow \mathbb{R}$ such that $M(p_f \in F) = 1$ and descends linearly to zero as

$$M(p) = \frac{R - d(p,F)}{R} \tag{7.2}$$

where $d(p,F)$ returns the point-to-set distance from pixel p to the closest point on the control curves $F = \cup F_i$. The target feature magnitude field $M' : I' \rightarrow \mathbb{R}$ is defined analogously.

The algorithm in Table 7.1 illustrates our algorithm for forming pixel clusters that serve as patches for synthesis. Feature matching is further improved when the candidate pool $P \subset F$ when $p'_0 \in F'$, and $P \cap F = \emptyset$ when $p'_0 \notin F'$.

Cluster Growth. We grow a cluster from p'_0 by integrating the target tangent field to construct a uniform arc-length sampling of the feature curve. We then create offset curves of this feature curve in both directions by propagating the feature curve in directions perpendicular to the tangent field.

We trace a feature curve from starting target pixel p'_0 by Euler integration of the tangent field,

$$p'_{k+1} = p'_k + T'(p'_k). \tag{7.3}$$

Let U' be the pixels of I' requiring texture synthesis.

While $U' \neq \emptyset$ ...

Let starting pixel $p'_0 = \text{argmax}_{p' \in U'} M'(p')$

(preferring neighbors to already textured pixels)

let $U' = U' \setminus \{p'_0\}$ and

set its preimage $p_0 = p'_0 - D(p'_0)$.

Let $P = \{p : ||p - p_0|| \leq r\}$ be an r-neighborhood of

candidates, where $r = 5$ works well.

Let bestCost $= \infty$.

For each $p \in P$ ...

Let cost = GrowCluster(p'_0, p)

if cost $<$ bestCost, then

bestCost = cost, and

$\hat{p} = p$

GrowCluster($p_0, \hat{p}$) and merge the result into I'.

Table 7.1: The synthesis process.

Note that T' is a unit length field and we work in screen coordinates with unit distance between pixels. We offset the sampled feature curves as

$$p'_{j+1,k} = p'_{j,k} + N'(p'_{j,k}) \tag{7.4}$$

for $j \geq 0$ where the normal field N' is constructed by rotating each vector of T' by 90°. A second offset is constructed in the opposite direction by

$$p'_{j-1,k} = p'_{j,k} - N'(p'_{j,k}) \tag{7.5}$$

for $j \leq 0$.

Euler integration accumulates the error in T' linearly, which can cause the sampling to deviate from the feature curve. Since we have the original feature curve from the user's specification, we can correct the uniform sampling with a reprojection onto the feature curve if its deviation grows too severe (tapering the correction on the previous samples). We also find it useful to smooth the cluster

$$p'_{j,k} = p'_{j,k} - \lambda \nabla^2 p'_{j,k} \tag{7.6}$$

using $\lambda = 0.7$ and several iterations to eliminate noise derived from T', N' and integration. We also restrain clusters from overlapping other feature lines.

We similarly construct the corresponding source image cluster $p_{j,k}$.

Cluster Parameterization. Ideally, the texture coordinates $c'(p'_{j,k}) = (j,k)$ which flattens each regularly sampled offset feature-curve cluster into a rectangle. The $p'_{j,k}$ are not necessarily located on integer pixel locations and their quantization is also a potential source of noise or aliasing. We thus sample the texture coordinates using a unit-radius filter weighted and normalized by the inverse of the distance to the query position.

We use this parameterization and filtered sampling to find a source image color at $p_{j,k}$ corresponding to the target cluster pixel $p'_{j,k}$. The cost of growing a cluster in I' from p'_0 is then measured as is usually done, by the average color difference in the overlapping area between the new cluster and the current synthesized texture of the target.

Scale Adaptive Clustering The deformation field D maps each target pixel in I' to a position in the source image I. A morph that compresses a large source area into a small target area can result in the blockiness artifacts shown in Fig. 7.3, because the start pixels of neighboring clusters in the compressed region of the target will map to non-proximate positions in the source with possibly irrelevant textures. Such cases will require a denser sampling.

We define a compression field $C : I' \rightarrow \mathbb{R}$ as the (discrete) Lipschitz constant of the deformation field, measured as

$$C(x,y) = \max ||D(x,y) - D(x \pm 1, y \pm 1)||. \tag{7.7}$$

where $C(x,y) = 1$ everywhere would indicate e.g. an incompressible flow. We clamp the compression field to values in $[1,3]$ to limit its effect on cluster size. Clusters are then constructed using parameters scaled by the multiplicative inverse of the compression value of the cluster's start pixel p'_0 thus adapting the scale of the cluster sampling to the contraction induced by the user-specified control curves.

7.4 Results

The user selects control lines manually using Lazy Snapping and deforms them with free-hand or spline curves. An interactive preview using normal morphing is provided for intuitive editing. Some control lines can be specified as passive. They are morphed by other control lines to maintain features in the result. In Fig. 7.1 (b), the silhouettes of the bunny's ears are morphed by the user, while the control lines inside each ear remain passive. The background in Fig. 7.1 (c) and (d) is excluded from synthesis with a user-selected matte, and is blended back into the result.

In Fig. 7.6 (b), a direct morphing on a photo of beach will not give satisfying result. Two control lines are used to morph the beach in Fig. 7.6 (c) and (d): one on the wave, one on the shadow of the wave. In Fig. 7.6 (d) the distance between wave and shadow is

Example	Time
Long bunny ear	88s
Bent bunny ear	73s
Fat bunny	221s
Beach, short shadow	257s
Beach, long shadow	241s
Wave	108s
Postdoc #1	218s
Postdoc #2	230s

Table 7.2: Running times for figures.

extended. In both results the resolution of the high frequency detail of the sand is slightly reduced. This is caused by the resampling of the source image at non-integer positions within each cluster and could be recovered by sharpening with histogram interpolation and matching [57].

Fig. 7.5 demonstrates a possible application of these techniques in the construction of caricatures. In these examples, regions are not only moved but enlarged requiring our method to synthesize hair and beard textures to fill in the gap. A matte was manually constructed to manage occlusions with the shirt.

Fig. 7.4 uses several control lines to make a smoothly breaking wave appear more torrential. The anisotropic textures of the rippling water and blowing foam are resynthesized and reoriented to match the new position of the feature curves.

Table 7.2 lists the running times for the figures shown. Running time depends on the searching radius, which was 5 pixels except for the beach example where it was 15 pixels. Run times were measured on a 3.40GHz Pentium 4 CPU.

7.5 Failure Cases

Our algorithm may fail if there are feature lines not marked by the user. Such feature lines might be broken after morphing. Although a search for the best match is performed for each patch during texture synthesis, its generally not sufficient to match all feature lines. A future work will be to make the feature line detection automatic.

This sun flower image in Fig. 7.14 contains many unmarked feature lines in the flower around the seeds. Several of them are broken in the result due to large morphing.

The bird example in Fig. 7.15 contains a texture with long lines. These lines are also not well aligned in the result. To fix this problem, we may need to incorporate the orientation of source texture into the tangent fields, in addition to interpolating feature orientation of user selected feature lines. The loss of sharpness due to resampling is also quite obvious on this high frequency texture.

7.6 Conclusion and Implementation Details

We have shown that the implementation of a feature-sensitive pixel clustering method leads to a useful texture synthesis tool that allows one to reshape an object's silhouette while preserving the detail of the surrounding texture.

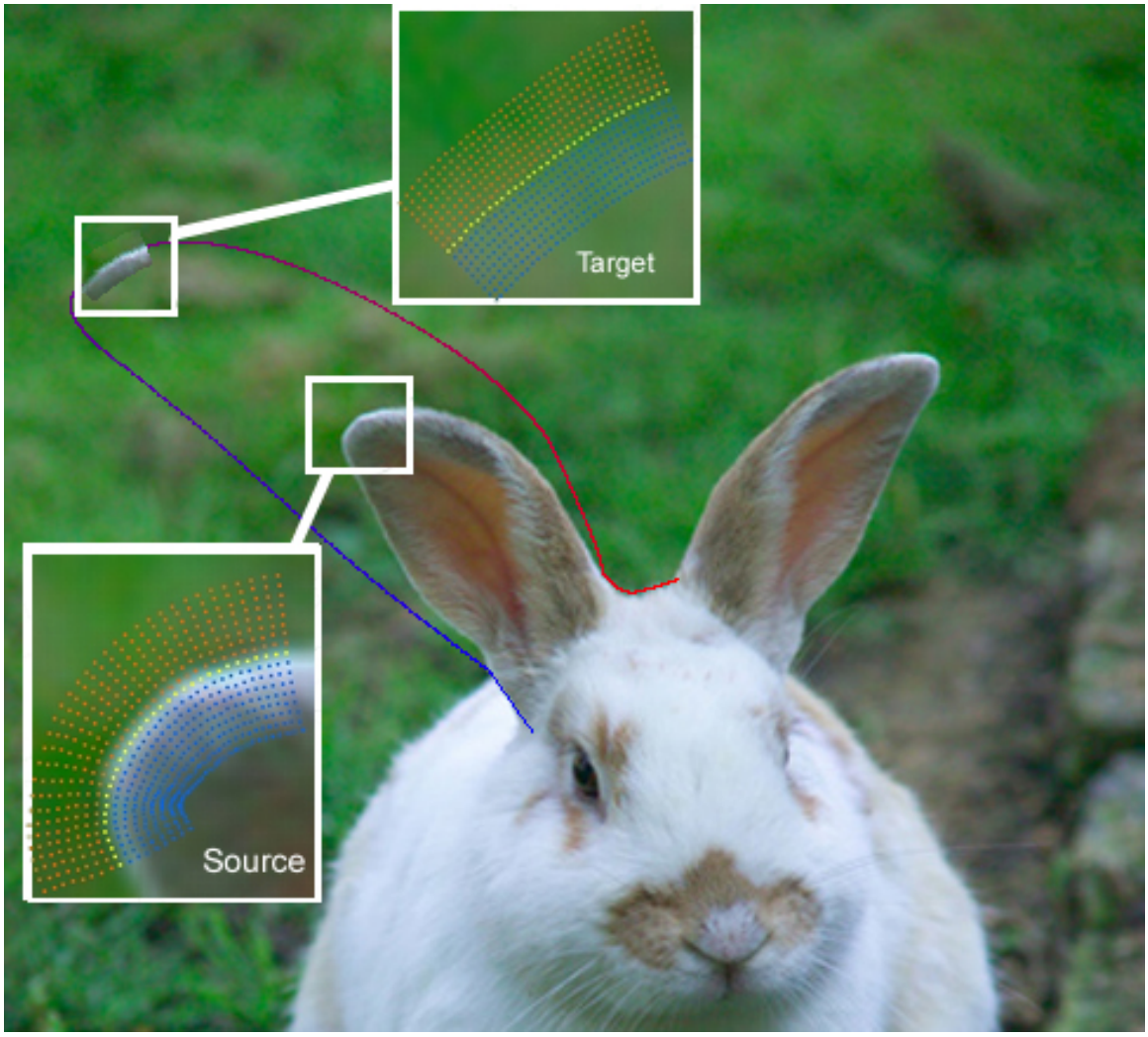
Because the parameterizations of the feature curves are arbitrary, one can encounter global orientation inconsistencies when constructing a single global tangent field. Our implementation actually computed a separate tangent field for each feature curve and curve tracing only references the tangent field corresponding to the closest feature curve.

To hasten execution time, we performed a Monte-Carlo search that considered only a random subset of neighboring pixels that nevertheless led to reasonable results.

We use Poisson image editing to seamlessly merge the new cluster with already synthesized part of the target image after applying Graphcut.

(a) (b) (c) (d) (e)

Figure 7.1: (a) A bunny's photo is scanned in. (b) Control curves are manually selected on the bunny's ears (c), dragged to new positions using the control curves and the ear interiors are resynthesized to accommodate the change. (d) A different ear style. (e) A bunny that can hardly stand on its own.

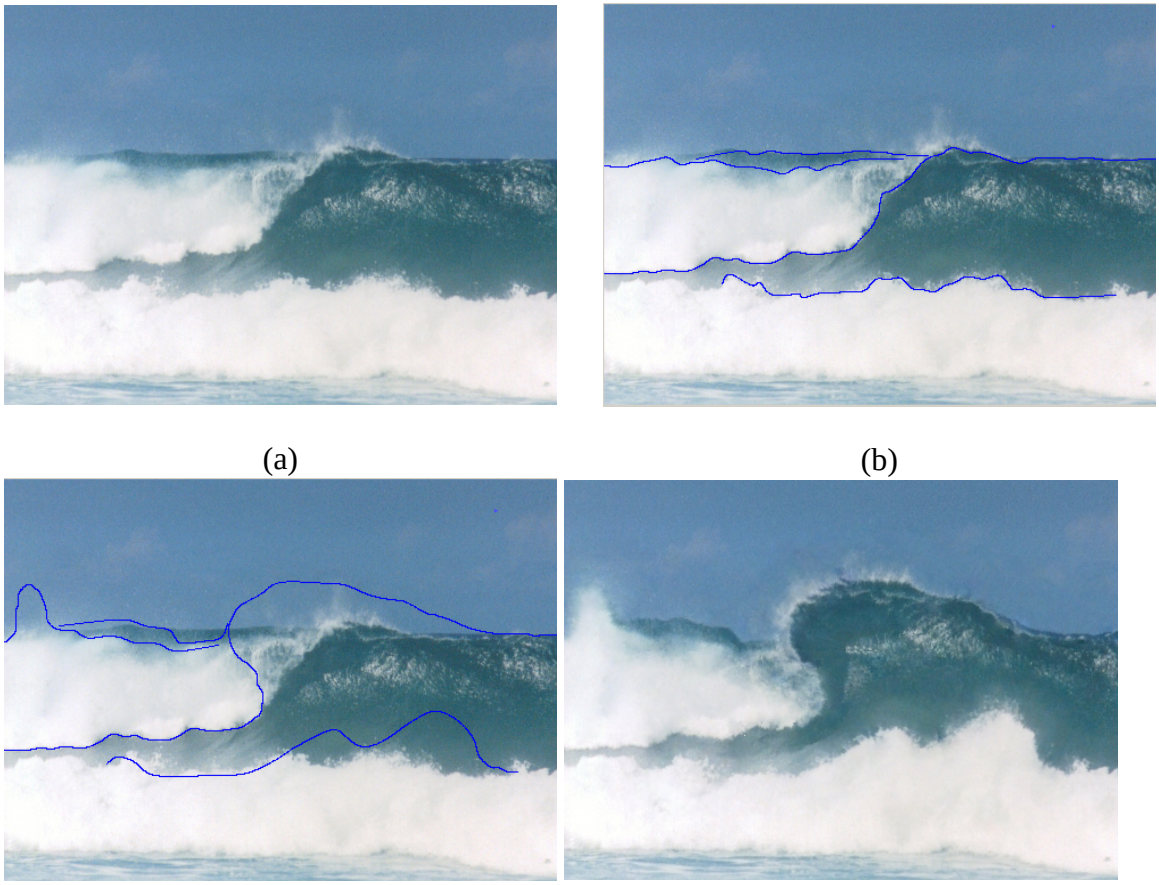


(a)

Figure 7.2: A source cluster on the bunny's ear is parameterized with the tangent field. The cluster line P'_k in feature direction is colored in yellow. Its texture is used to generate a target cluster with a feature curve of different shape.

(a) (b) (c)

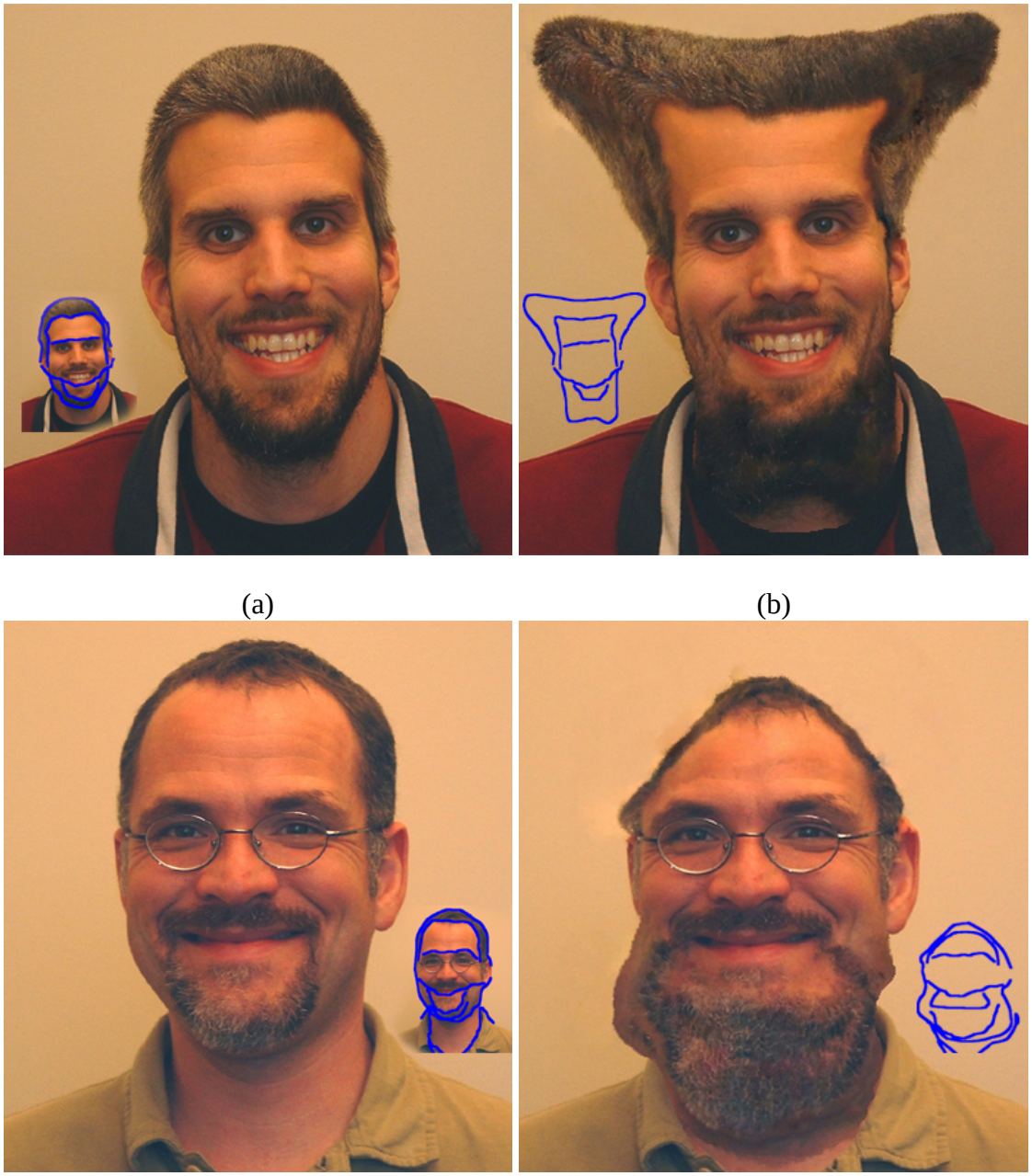
Figure 7.3: Compression field for a deformed wave (a) caused block artifacts in (b). It is reduced in (c) by using adaptive cluster size.



(a) (b)

(c) (d)

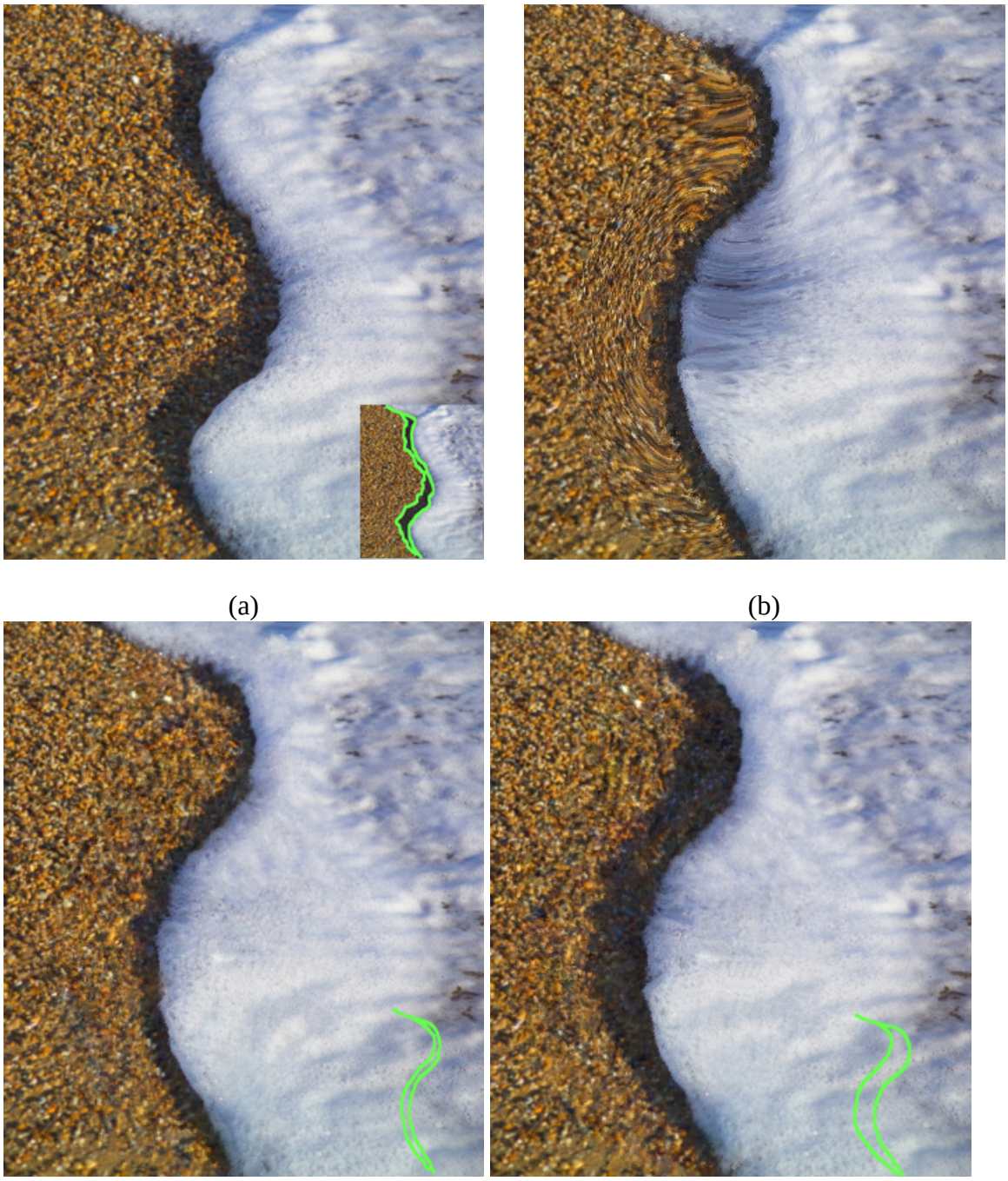
Figure 7.4: (a) A photo of an ocean. (b) Several waves are manually selected. (c) The waves are morphed by the user. (d) The result is a raging ocean.



(a) (b)

(c) (d)

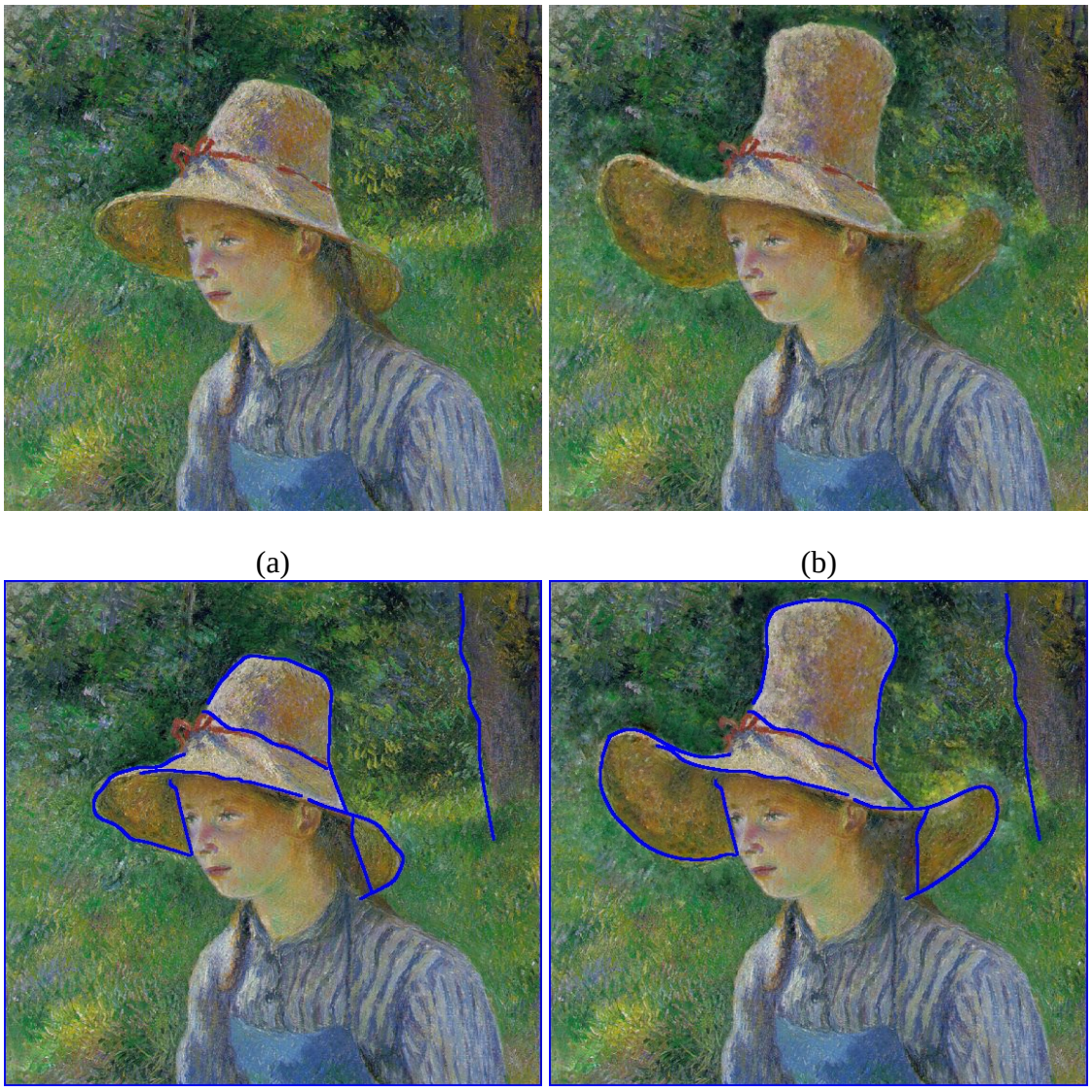
Figure 7.5: A face is morphed.



(a) (b)

(b) (b)

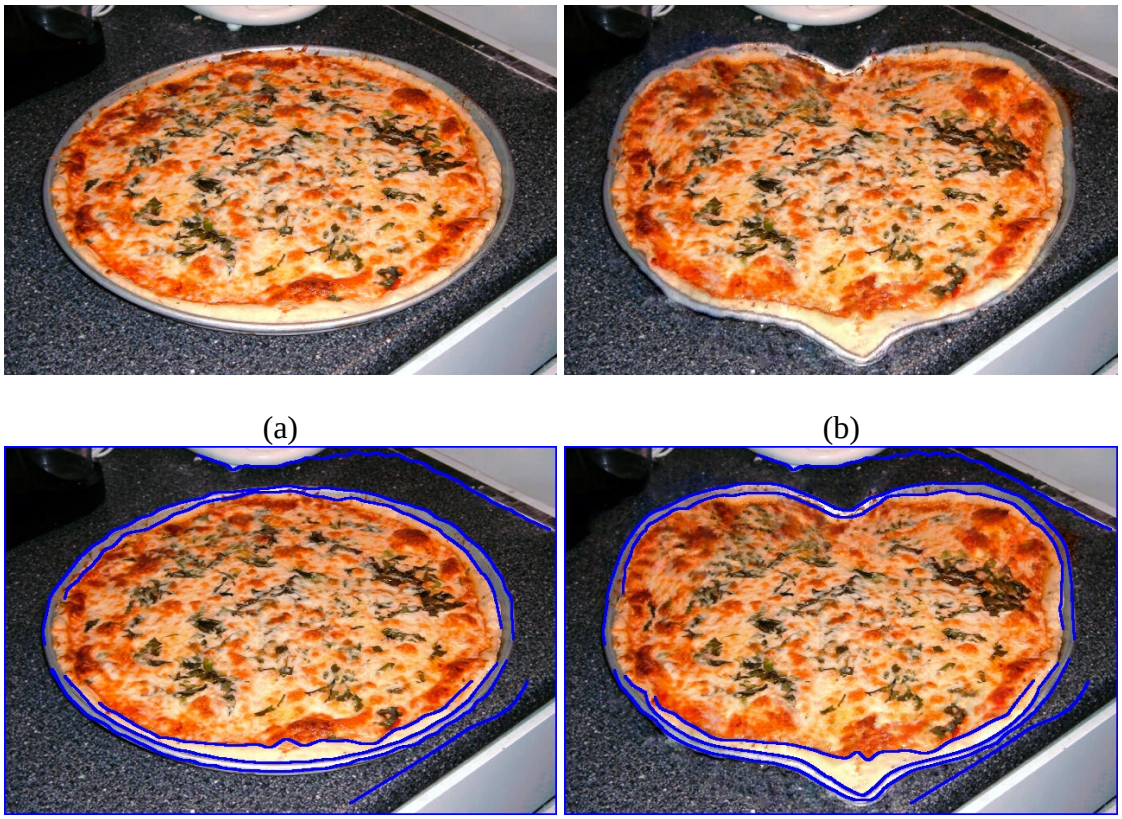
Figure 7.6: A simple morph of a photo of beach (a) will yield unsatisfied result (b). Our approach gives much better morphing effect in (c). The shadow range of the wave is extended in (d).



(a) (b)

(c) (d)

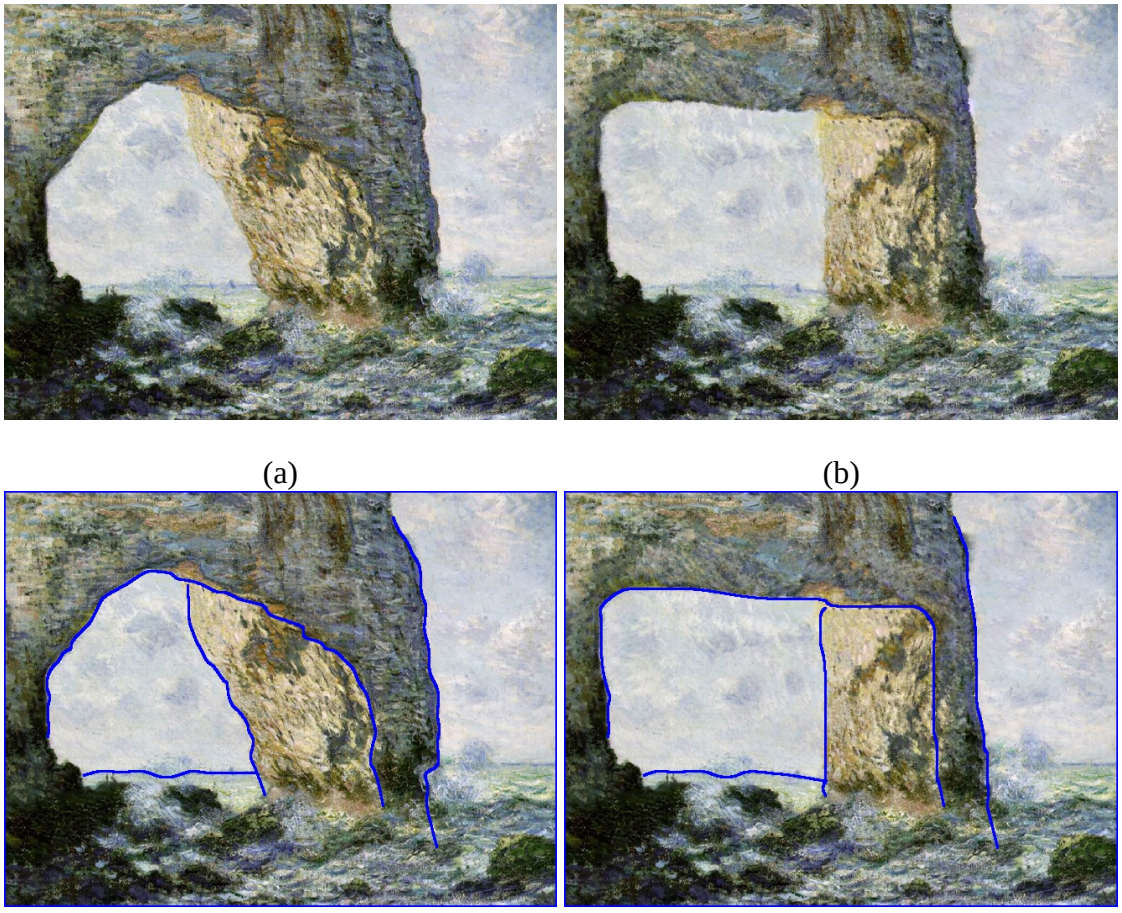
Figure 7.7: An image (a) is morphed (b) by deforming control curves from (c) to (d).



(a) (b)

(c) (d)

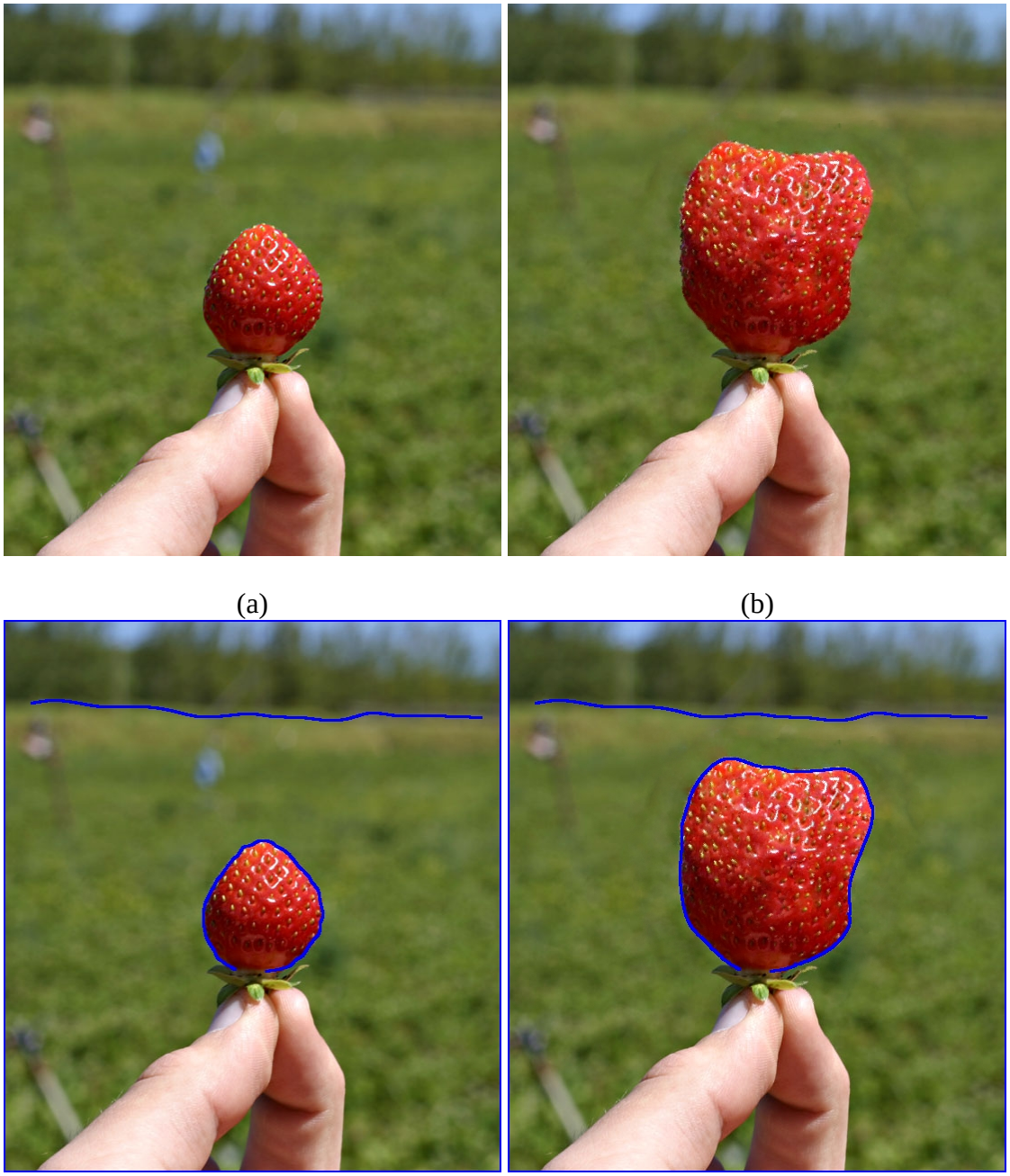
Figure 7.8: An image (a) is morphed (b) by deforming control curves from (c) to (d).



(a) (b)

(c) (d)

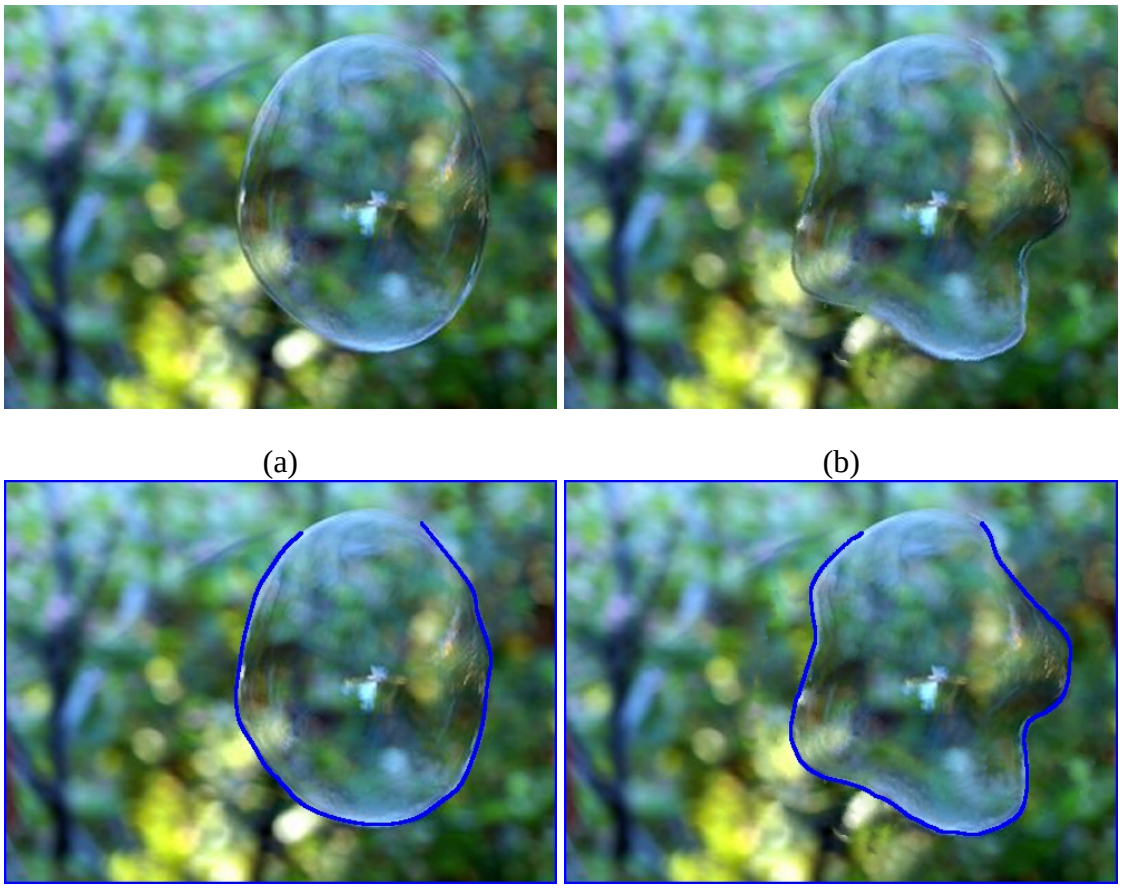
Figure 7.9: An image (a) is morphed (b) by deforming control curves from (c) to (d).



(a) (b)

(c) (d)

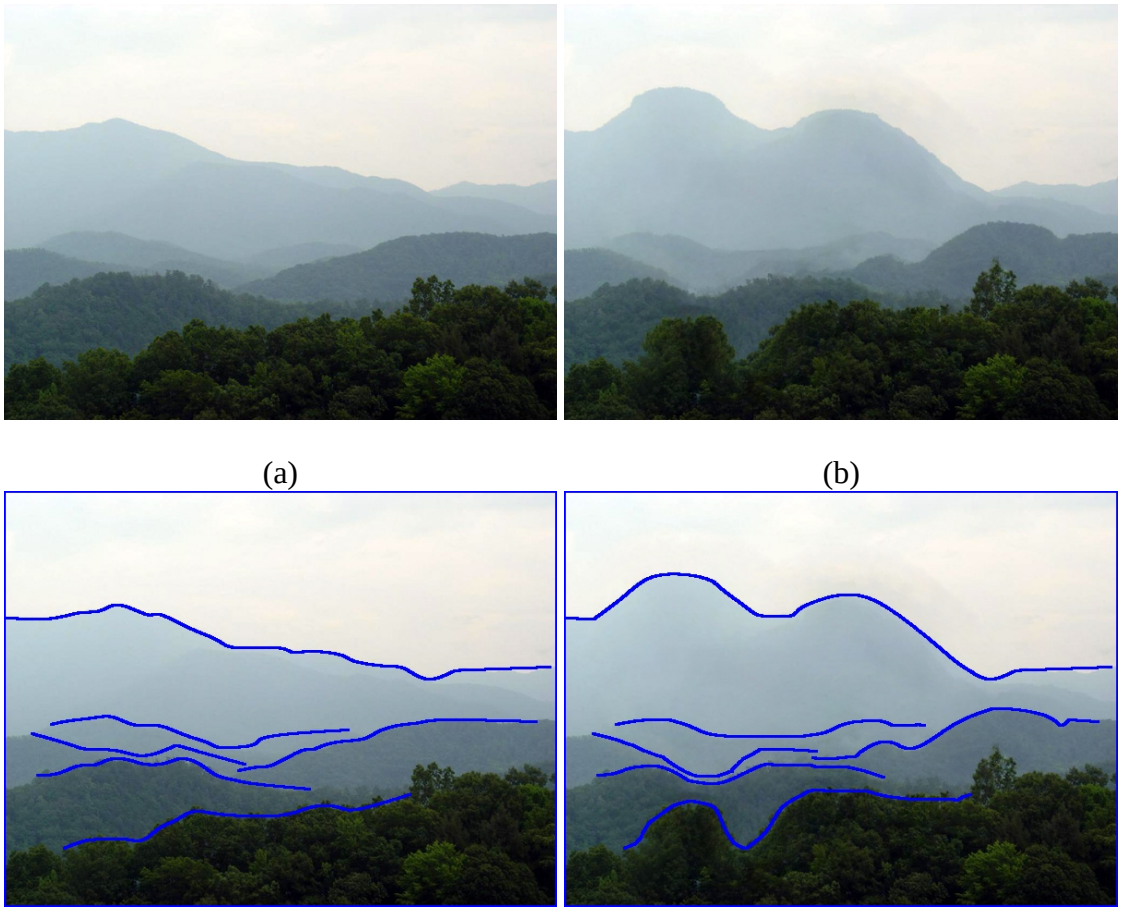
Figure 7.10: An image (a) is morphed (b) by deforming control curves from (c) to (d).



(a) (b)

(c) (d)

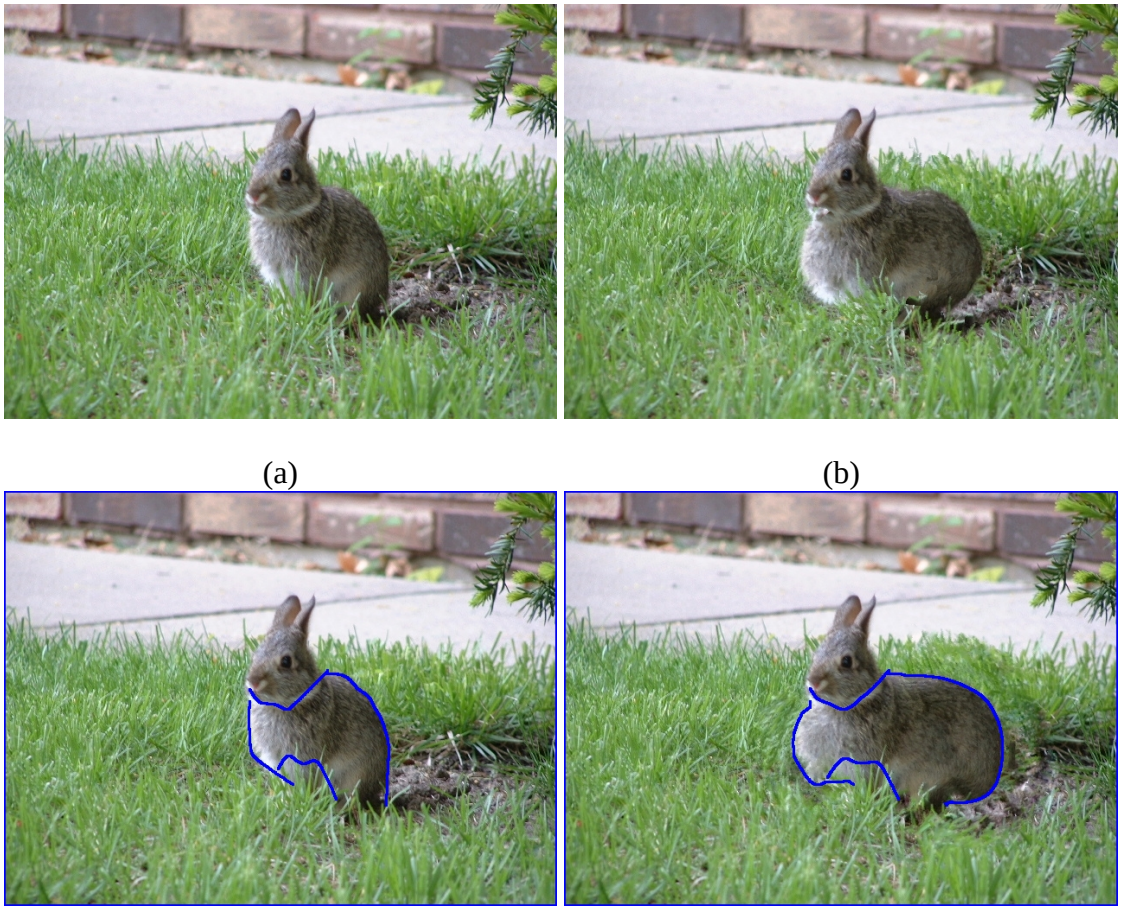
Figure 7.11: An image (a) is morphed (b) by deforming control curves from (c) to (d).



(a) (b)

(c) (d)

Figure 7.12: An image (a) is morphed (b) by deforming control curves from (c) to (d).



(a) (b)

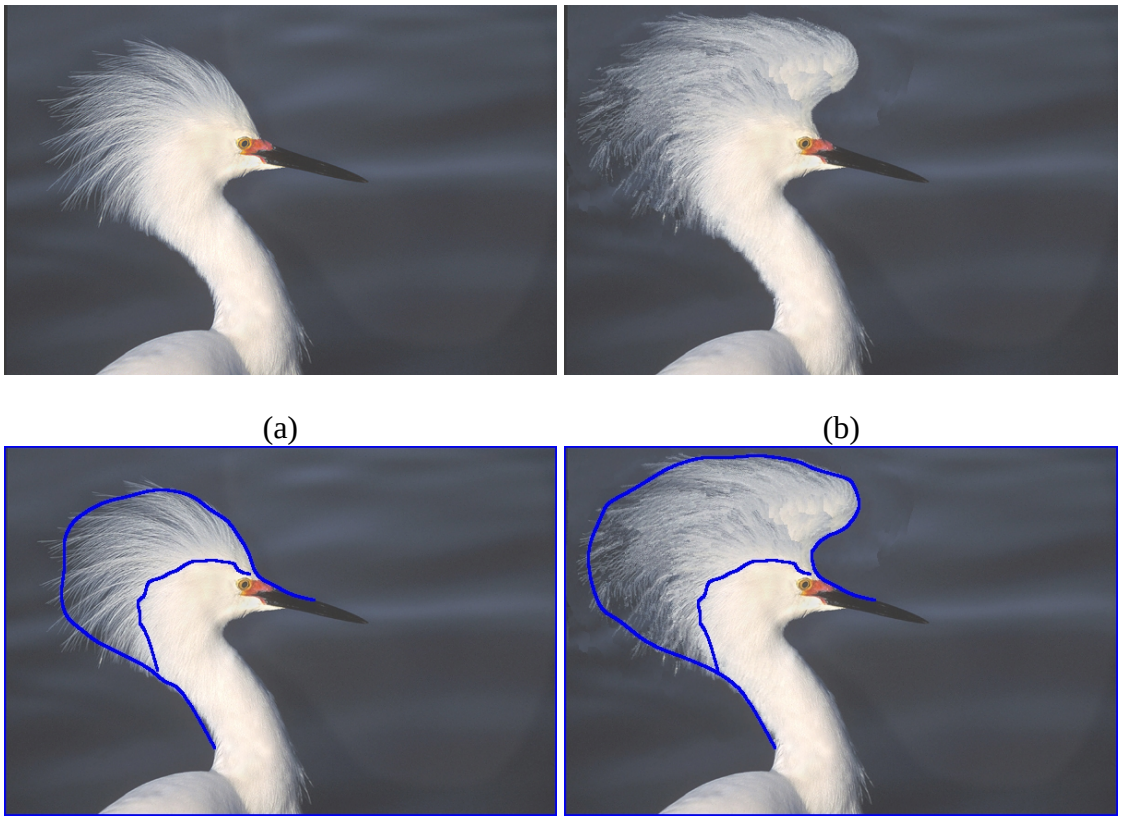
(c) (d)

Figure 7.13: An image (a) is morphed (b) by deforming control curves from (c) to (d). Artifacts in the background can be removed if the user specify a matte for the foreground and exclude the background from synthesis.

(a) (b)

(c) (d)

Figure 7.14: Failure case with a sun flower.



(a) (b)

(c) (d)

Figure 7.15: Failure case with a bird.

References

[1] G. Turk. "Texture synthesis on surfaces." *Proc. SIGGRAPH 2001* (2001).

[2] L.-Y. Wei and M. Levoy. "Texture synthesis over arbitrary manifold surfaces." *SIGGRAPH 2001* (2001).

[3] C. A. J. and M. J. E. "A mathematical introduction to fluid mechanics." *Springer-Verlag. Texts in Applied Mathematics 4. Second Edition., New York* (1990).

[4] J. Stam. "Stable fluids." *SIGGRAPH 99 Conference Proceedings, Annual Conference Series, pages 121-128* (1999).

[5] R. Fedkiw, J. Stam, and H. Jensen. "Visual simulation of smoke." *SIGGRAPH 2001 Conference Proceedings, pages 21-30* (2001).

[6] B. Heckel, G. Weber, B. Hamann, and K. I. Joyand. "Construction of vector field hierarchies." *IEEE Visualization '99* (1999).

[7] H. Garcke, T. Preuer, R. M., A. Telea, U. Weikard, and J. v. Wijk. "A continuous clustering method for vector fields." *IEEE Visualization '2000* (2000).

[8] H. Weimer and J. Warren. "Subdivision schemes for fluid flow." *SIGGRAPH 1999, Computer Graphics Proceedings* (1999).

[9] G. Wallace. "The jpeg still picture compression standard." *Communications of the ACM, 34(4):30-44* (1991).

[10] B.-L. Yeo and B. Liu. "Volume rendering of dct-based compressed 3d scalar data." *IEEE Transactions on Visualization and Computer Graphics, Volume 1, pages 29-43* (1995).

[11] G. Taubin. "Geometric signal processing on polygonal meshes." In *State of the Art Report*, pp. 81–96. Eurographics (2000).

[12] J. E. Fowler and R. Yagel. "Lossless compression of volume data." *Proceedings of the 1994 Symposium on Volume Visualization, Washington, DC, October 1994, pp. 43-53* (1994).

[13] D. L. Gall. "MPEG: A video compression standard for multimedia applications." *Communications of the ACM, 34(4):46–58, April 1991* (1991).

[14] W. S. Stefan Guthe. “Real-time decompression and visualization of animated volume data.” *Proceedings of Visualization 2001* (2001).

[15] M. Roth and R. Peikert. “A higher-order method for finding vortex core lines.” *Proceedings of the conference on Visualization ’98* (1998).

[16] N. D. and C. C. “Lossless region of interest coding.” *Signal Processing, Vol. 78, No. 1, pp. 1-17* (1999).

[17] B. Grinstead, H. Sari-Sarraf, S. Gleason, and S. Mitra. “Preliminary validation of content-based compression of mammographic images.” *SPIE Medical Imaging: Image Processing 2001,San Diego, CA* (2001).

[18] B.-S. Sohn, C. Bajaj, and V. Siddavanahalli. “Feature based volumetric video compression for interactive playback.” *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics, Boston, Massachusetts* (2002).

[19] U. Pinkall and K. Polthier. “Computing discrete minimal surfaces and their conjugates.” *Exp. Math.* **2** 15–36 (1993).

[20] T. Duchamp, A. Certain, A. DeRose, and W. Stuetzle. “Hierarchical computation of PL harmonic embeddings.” (1997). Preprint.

[21] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. “Implicit fairing of irregular meshes using diffusion and curvature flow.” In *Proc. SIGGRAPH*, pp. 317–324 (1999).

[22] Z. Karni and C. Gotsman. “Spectral compression of mesh geometry.” In *Proc. SIGGRAPH*, pp. 279–286 (2000).

[23] M. S. Floater. “Mean value coordinates.” *Computer Aided Geometric Design* **20** 19–27 (March 2003).

[24] B. K. Horn. “Height and gradient from shading.” *International journal of computer vision, 5:1,, 37-75, 1990* (1990).

[25] V. Kwatra, A. Schoedl, I. Essa, G. Turk, and A. Bobick. “Graphcut textures: Image and video synthesis using graph cuts.” *Proc. SIGGRAPH 2003* (2003).

[26] A. McNamara, A. Treuille, Z. Popovic, and J. Stam. “Keyframe control of smoke simulations.” *Proc. SIGGRAPH 2003* (2003).

[27] P. Perez, M. Gangnet, and A. Blake. “Poisson image editing.” *Proc. SIGGRAPH 2003* pp. 313 – 318 (2003).

[28] I. Drori, D. Cohen-Or, and H. Yeshurun. “Fragment-based image completion.” *Proc. SIGGRAPH 03* pp. 303–312 (2003).

[29] S. Zelinka and M. Garland. “Towards real-time texture synthesis with the jump map.” *Proceedings of the Thirteenth Eurographics Workshop on Rendering Techniques (2002), Eurographics Association,* (2002).

[30] S. Zelinka and M. Garland. "Interactive texture synthesis on surfaces using jump maps." *Proceedings of the Eurographics Symposium on Rendering 2003, pages 90C96, Leuven, Belgium* (2003).

[31] H. Fang and J. C. Hart. "Textureshop: Texture synthesis as a photograph editing tool." *Proc. SIGGRAPH 2004, Los Angeles, California* (2004).

[32] Y. Liu, W.-C. Lin, and J. Hays. "Near-regular texture analysis and manipulation." *TOG* **23** 368–376 (2004). (Proc. SIGGRAPH).

[33] M. Pollefeys, L. V. Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch. "Visual modeling with a hand-held camera." *Intl. J. Comp. Vision* **59** 207–232 (2004).

[34] R. Zhang, P.-S. Tsai, J. E. Cryer, and M. Shah. "Shape from shading: A survey." *IEEE Transactions on Pattern Analysis and Machine Intelligence* **21** 690–706 (1999).

[35] L. Zhang, G. Dugas-Phocion, J.-S. Samson, and S. Seitz. "Single view modeling of free-form scenes." *Proc. CVPR* (2001).

[36] L. Zhang, B. Curless, and S. M. Seitz. "Spacetime stereo: Shape recovery for dynamic scenes." *Proc. CVPR* pp. 367–374 (2003).

[37] L. Zhang, N. Snavely, B. Curless, and S. M. Seitz. "Spacetime faces: High resolution capture for modeling and animation." *Proc. SIGGRAPH 2004* pp. 548–558 (2004).

[38] L. Torresani, D. Yang, G. Alexander, and C. Bregler. "Tracking and modeling non-rigid objects with rank constraints." *Proc. IEEE Computer Vision and Pattern Recognition, 2001* (2001).

[39] M. Brand. "Morphable 3d models from video." *Proc. IEEE Computer Vision and Pattern Recognition, 2001* (2001).

[40] C. Bregler, A. Hertzmann, and H. Biermann. "Recovering non-rigid 3d shape from image streams." *Proc. IEEE Computer Vision and Pattern Recognition, 2000* (2000).

[41] M. J. Black and P. Anandan. "The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields." *Computer Vision and Image Understanding, CVIU, 63(1), pp. 75-104, Jan. 1996.* (1996).

[42] D. DeCarlo and D. Metaxes. "Optical flow constraints on deformable models with applications to face tracking." *Intl. J. of Comp. Vision* **38** 99–127 (July 2000).

[43] D. DeCarlo and D. Metaxes. "Adjusting shape parameters using model-based optical flow residuals." *IEEE Trans. on PAMI* **24** 814–823 (2002).

[44] S.-Y. Lee, K.-Y. Chwa, S. Y. Shin, and G. Wolberg. "Image metamorphosis using snakes and free-form deformations." *Proc. SIGGRAPH 1995, Los Angeles, California* (1995).

[45] A. A. Efros and W. T. Freeman. “Image quilting for texture synthesis and transfer.” *SIGGRAPH 2001* (2001).

[46] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum. “Lazy snapping.” *Proc. SIGGRAPH 2004, Los Angeles, California* (2004).

[47] Y.-Y. Chuang, A. Agarwala, B. Curless, D. H. Salesin, , and R. Szeliski. “Video matting of complex scenes.” *Proc. SIGGRAPH 2002, San Antonio, Texas* (2002).

[48] Y. Li, J. Sun, and H.-Y. Shum. “Video object cut and paste.” *Proc. SIGGRAPH 2005, Los Angeles, California* (2005).

[49] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. “Image analogies.” *Proc. SIGGRAPH 2001* (2001).

[50] Q. Wu and Y. Yu. “Feature matching and deformation for texture synthesis.” *Proc. SIGGRAPH 2004* (2004).

[51] D. J. Heeger and J. R. Bergen. “Pyramid-based texture analysis/synthesis.” *Proc. SIGGRAPH 1995* (1995).

[52] L.-Y. Wei and M. Levoy. “Fast texture synthesis using tree-structured vector quantization.” *Proc. SIGGRAPH 2000* (2000).

[53] S. Brooks, M. Cardle, and N. Dodgson. “Concise user control for texture-by-numbers cloning.” *SIGGRAPH Technical Sketch 2003* (2003).

[54] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra. “Texture optimization for example-based synthesis.” *Proc. SIGGRAPH 2005* (2005).

[55] J. Sun, L. Yuan, J. Jia, and H.-Y. Shum. “Image completion with structure propagation.” *Proc. SIGGRAPH 2005* (2005).

[56] W. A. Barrett and A. S. Cheney. “Object-based image editing.” *Proc. SIGGRAPH 2002* (2002).

[57] W. Matusik, M. Zwicker, and F. Durand. “Texture design using a simplicial complex of morphable textures.” *Proc. SIGGRAPH 2005* (2005).

Author’s Biography

Hui Fang received the bachelor’s degree in physics from Nanjing University, China and the master’s degree in computer science from University of Illinois at Urbana Champaign (UIUC). He is current a PhD candidate in UIUC and will receive his PhD degree in May 2006. His research interests include computer graphics and vision, with a focus on image-based modeling and rendering. He served as a reviewer for numerous conferences and journals in the field.